Université de Montréal

Analyse de la qualité du logiciel: Une approche par visualisation et simulation

Par

Mohamed Rouatbi

Département d'Informatique et de Recherche Opérationnelle
Faculté des Arts et des Sciences

Mémoire présenté à la Faculté des Études Supérieures
en vue de l'obtention du grade de
Maître ès Sciences (M.Sc.)
en Informatique

Aout, 2005

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Analyse de la qualité du logiciel:
Une approche par visualisation et simulation

Présenté par :

Mohamed Rouatbi

Membre du jury

Pierre Poulin
Président-rapporteur

Houari A. SAHRAOUI
Directeur de recherche

KÉGL Balázs
Codirecteur de recherche

Philippe Langlais
Membre du jury

Mémoire accepté le 14 novembre 2005

Sommaire

L'exploitation des différentes techniques de visualisation et de simulation est le concept clef dans cette recherche. Ces techniques qui ont déjà fait leurs preuves dans plusieurs domaines nous permettent de concilier deux types de modèles d'évaluation de la qualité du logiciel. Ces deux modèles sont les modèles abstraits qui sont basés sur des données historiques et les modèles experts.

Notre contribution majeure est la proposition d'une approche de simulation des phénomènes logiciels en se basant sur les modèles et les théories empruntés à d'autres domaines. Nous illustrons cette position à travers l'exemple de l'adaptation du modèle de propagation de chaleur en thermodynamique à la modélisation et la simulation de la propagation des effets de changements dans les programmes à objets.

Mots clefs :

Impact du changement, visualisation, qualité du logiciel, simulation, visualisation du logiciel.

Abstract

Exploitation of various techniques of visualization and simulation is the key concept in this research. These techniques which already have proven their effectiveness in several fields allow us to reconcile two types of software quality assessment models. These two models are the abstract models which are based on historical data and the expert models.

Our major contribution is the proposal of a simulation approach of software phenomena based on the models and the theories borrowed from other fields. We illustrate this position through the example about the adaptation of the heat propagation model in thermodynamics to simulate change impact in object oriented programs.

Keywords:
Change impact, simulation, visualization, software quality, software visualization and simulation.

Table des matières

Liste des figures

À ma mère

Chapitre 1
Introduction

1.1 Contexte et problématique

Nous sommes dans une période où l'utilisation des logiciels a connu une prolifération exponentielle. Ces derniers occupent désormais de plus en plus de place dans notre vie. Les applications des logiciels sont très variées et sont souvent nécessaires pour assurer le bon fonctionnement de certains services essentiels. Nous citons notamment la distribution de l'électricité, la gestion des appels d'urgence, le contrôle des appareillages médicaux, etc.

Face à l'avancement technologique, aux changements des besoins et des spécifications, on a besoin d'adapter les logiciels rapidement et efficacement. Ces adaptations impliquent souvent deux alternatives. La première consiste à se procurer un nouveau logiciel répondant aux nouvelles attentes et conforme aux nouveaux changements. La deuxième consiste à effectuer des modifications au présent logiciel. Dans les deux cas il y a une nécessité d'évaluation la qualité du logiciel acheté ou développé. En effet, on a besoin non seulement d'évaluer la qualité du logiciel (composantes logicielles) mais aussi de la gérer, de la surveiller, de la contrôler et de l'améliorer durant toute sa durée de vie. D'autant plus qu'il est prouvé que le contrôle de la qualité permet de faire des économies substantielles spécialement au niveau du coût d'entretien qui représente souvent plus de 50 % du budget de développement du logiciel.

Aujourd'hui il existe essentiellement deux types de modèle d'évaluation de la qualité du logiciel : les modèles experts et les modèles abstraits à partir de données historiques. Les modèles experts sont des modèles qui sont basés sur des heuristiques souvent reliées à un certain domaine d'application. Ces derniers requièrent une vérification « manuelle » du code source ou des différents artefacts logiciels. Ainsi, un expert d'après son expérience, certaines règles et patrons préconçus, pourrait juger de la qualité du logiciel. Les modèles abstraits qui sont basés sur des données historiques sont des modèles qui ont un fondement automatique. C'est-à-dire qu'en utilisant certaines formules et méthodes nous pouvons, à partir de ces données historiques, évaluer et éventuellement prédire la qualité du logiciel.

Bien que ces deux modèles d'évaluation soient couramment utilisés, il n'en demeure pas moins qu'ils souffrent de quelques lacunes. Le plus grand problème pour les modèles basés sur les données historiques est le manque de données. En effet, ces dernières ne sont pas collectées d'une manière systématique pendant tout le long du cycle de vie du logiciel. Pour les modèles experts la validation et la reconnaissance sont les principaux ennuis. Donc, notre approche consiste à combiner les deux types de modèle en utilisant des techniques de visualisation.

L'exploitation des différentes techniques de visualisation est le concept clef dans cette recherche. Ces techniques qui ont déjà fait leurs preuves dans plusieurs domaines nous permettant de concilier les deux modèles d'évaluation de la qualité du logiciel. Ainsi, les méthodes quantitatives qui sont basées sur l'utilisation des métriques nous fournissent les informations nécessaires pour décrire et présenter les entités logicielles. Alors que les modèles experts nous assistent dans la tâche d'interprétation et la compréhension des présentations produites.

1.2 Objectifs

L'objectif de cette étude est le développement d'une plateforme de visualisation et de simulation générique qui permet de : (1) comprendre et faciliter l'analyse et l'évaluation de la qualité du logiciel et (2) étudier certains phénomènes logiciels.

1.3 Contributions

Les contributions majeures de ce mémoire sont les suivantes :

- Présentation de l'état de l'art et de l'utilisation de la visualisation dans le domaine de génie logiciel.

- Introduction d'une approche de modélisation et simulation des phénomènes liés au logiciel en adaptant les modèles théoriques issus d'autres domaines.

- Développement d'un logiciel de simulation (ANIMETRIX).

- La mise en pratique de l'approche proposée en utilisant un modèle de propagation de la chaleur dans les systèmes physiques comme point de départ à la modélisation et la simulation de la propagation des effets de changements.

1.4 Contenu

Le chapitre 2 est une introduction aux notions de visualisation et de perception. Dans les deux premières sections nous présentons brièvement le système de perception et de vision humaine. Comprendre le fonctionnement, les capacités et les limites de tels systèmes constitue une étape nécessaire pour le développement d'outils de visualisation utilisables et

efficaces. Dans la dernière section nous présentons un survol des notions de stimulation et de contrôle de l'attention.

Le chapitre 3 est un aperçu sur l'état de l'art de l'application de la visualisation dans le domaine du génie logiciel. Nous commençons par introduire la notion de visualisation de l'information qui est sans doute le fondement de l'application de la visualisation dans plusieurs domaines y compris celui du logiciel. Ensuite, nous nous intéressons à la classification de la visualisation du logiciel. Une telle classification nous permettra, entre autre, de bien définir notre système de visualisation ainsi que de se positionner par rapport aux autres travaux fais dans ce domaine et qui sont présentés dans le reste des sections de ce chapitre.

Le chapitre 4 commence par la présentation de l'apport de la visualisation dans l'évaluation et l'estimation de la qualité du logiciel. Après, nous présentons les étapes nécessaires à toute visualisation qui sont la normalisation et l'association. En même temps, nous exposerons notre premier prototype de visualisation qu'on a nommé MDS.

Le chapitre 5 présente notre approche de simulation des phénomènes logiciels en se basant sur les modèles et les théories empruntés à d'autres domaines. Nous y présentons les différentes étapes requises pour la construction de telle simulation.

Dans le chapitre 6 nous allons étudier le phénomène d'impact du changement dans le domaine logiciel en utilisant le modèle de propagation de la chaleur tiré du domaine de la science physique. Par ailleurs, nous allons utiliser notre plateforme ANIMETRIX que nous avons développée pour faciliter les tâches d'association et de simulation.

Chapitre 2
La visualisation et la perception

Ce chapitre est une introduction aux notions de visualisation et de perception. Dans les deux premières sections nous présentons brièvement le système de perception et de vision humaine. Comprendre le fonctionnement, les capacités et les limites de tels systèmes, constitue une étape nécessaire pour le développement d'outils de visualisation utilisables et efficaces. Dans la dernière section nous présentons un survol des notions de stimulation visuelle et de contrôle d'attention.

2.1 La visualisation

La visualisation est l'habileté d'utilisation de « l'œil de l'esprit » pour visualiser et conceptualiser les pensées et les idées. Désignée souvent comme l'imagerie mentale, elle est reconnue comme une composante importante de la mémoire et de la créativité. Elle est intimement liée à la capacité de lecture, de compréhension, de prononciation et d'écriture et aux habiletés de performance symbolique [12].

Cette définition met surtout l'emphase sur l'importance de la visualisation dans la vie de tous les jours ainsi que sur son rôle dans le processus d'apprentissage. En effet, on dit souvent « je vois » pour signifier qu'on a compris de quoi il s'agit, ou même on dit « faire la lumière sur un problème » pour signifier qu'on va se concentrer sur ce problème particulier pour le comprendre [1]. Donc, la visualisation est un point central dans le processus cognitif humain.

2.2 Buts de l'utilisation de la visualisation

Les applications basées sur la visualisation sont de plus en plus utilisées et ce dans de nombreux domaines. En médecine, par exemple, on utilise l'imagerie médicale pour repérer des pathologies ou pour vérifier le bon fonctionnement du cœur. En météorologie, la visualisation des images et des données captées par satellite facilite leur traitement et analyse. Un autre exemple montré par la figure 1 est celui du domaine boursier où les illustrations qui regroupent plusieurs indices boursiers sont des outils incontournables.

Cette prolifération d'outils de visualisation est due aux grands succès qu'ils ont eus dans ces domaines. Par ailleurs, les représentations graphiques produites par ces derniers offrent un support à la prise de décision et à la recherche d'explications. De plus, elles donnent une vue d'ensemble permettant de détecter la présence de régularités caractéristiques de certains phénomènes. Finalement, elles permettent de distinguer les catégories, les structures, les regroupements et les données extrêmes. En résumé, la visualisation permet de détecter le sens des données en exploitant le système visuel humain.

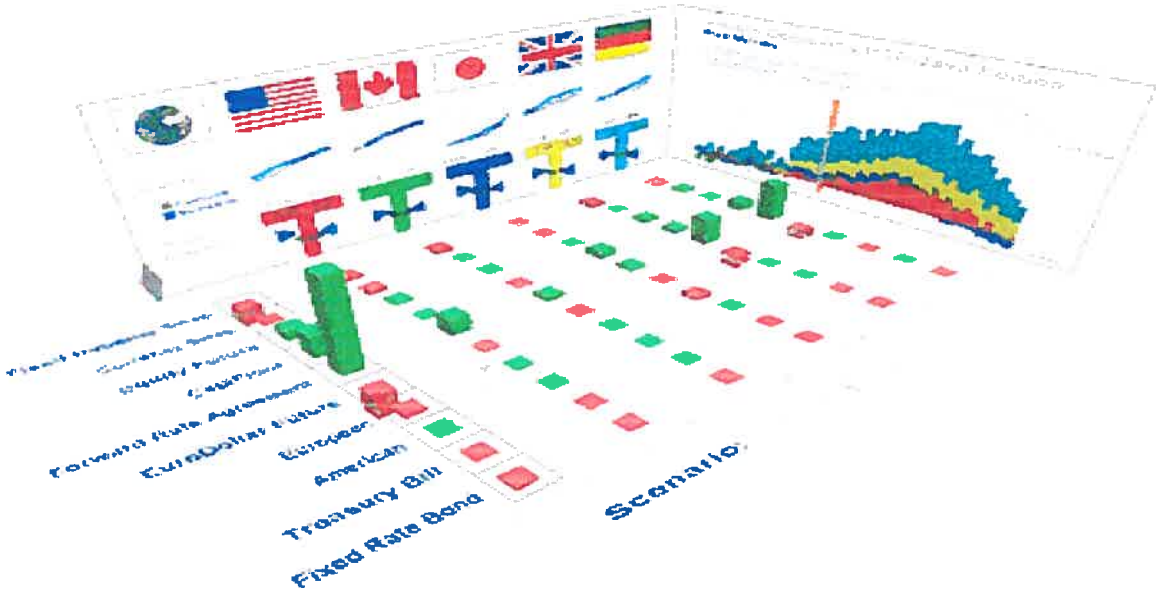


Figure 1 Visualisation de plusieurs d'indices boursiers dans une même localisation [14].

2.3 La perception

La perception est la première phase dans le processus de visualisation. Elle est primordiale. Sans elle la visualisation ne peut avoir lieu. C'est en fait le premier contact avec le monde extérieur. D'ailleurs, le processus de perception pour l'être humain est analogue à celui dans le monde de l'ordinateur, d'interception et de traitement des signaux émanant des périphériques d'entrées comme la souris, le clavier, le digitaliseur, la caméra, etc.

D'une manière générale, la perception est la reconnaissance et l'interprétation des stimuli sensoriels [15]. Elle est basée sur un système de traitement sophistiqué et complexe qui est conçu pour extraire d'une façon efficace l'information de notre environnement [17]. Des exemples de stimuli sont : les odeurs que nous percevons avec l'odora, les textures et la température que nous percevons avec le toucher, les formes et les couleurs que nous percevons avec les yeux, etc.

Toutefois, le système de perception humaine n'est pas infaillible. Il a des capacités limitées dont il faut tenir compte pour développer des systèmes de visualisation efficaces et utilisables. En effet, compte tenu de l'énorme quantité d'informations reçues de l'environnement, le système de perception humaine effectue un certain filtrage et ignore certains détails qui sont jugés non importants dans certains contextes. De plus, la capacité de la perception est limitée par l'aptitude et la capacité des récepteurs sensoriels comme c'est le cas de l'ultrason qu'on ne peut entendre ou des lumières infrarouges que l'on ne peut voir, etc.

Dans le cadre de cette étude, nous nous intéressons seulement à la perception par la vision et ceci essentiellement pour deux raisons. Premièrement, les études en sciences cognitives dont celles de Miller [20] affirment que la capacité de perception de l'être humain est d'autant plus grande lorsque les habiletés visuelles sont utilisées. D'après l'expérience de Pollack [61], on est capable de distinguer sans confusion seulement six tonalités différentes (voir figure 2). Par contre, comme le montre l'expérience de Hake et Garner [62] en

utilisant les capacités visuelles, telle que la différenciation de localités distinctes d'un marqueur sur une ligne, on discerne sans confusion de dix à quinze localités. Deuxièmement, il est démontré que la visualisation par la vision augmente notre capacité de résolution de problèmes et de traitement de données unies ou multidimensionnelles.

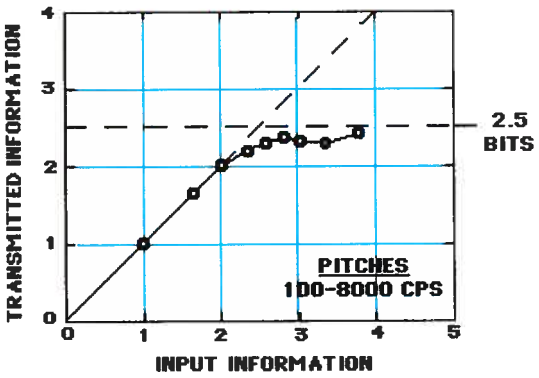


Figure 2 Courbe représentant le nombre de tonalités identifiées [20].

L'axe des X représente la quantité d'informations émises (dans ce cas ci les différentes tonalités) en bit exemple : 1 bit représente 2 différentes tonalités, 2 bits représentent 4 tonalités, etc. L'axe des Y représente la quantité d'information transmise, en d'autres mots le nombre de tonalités que la personne a distingué sans confusion. On remarque ainsi que la capacité maximale est à 2.5 c'est-à-dire qu'on est capable de discerner sans confusion 6 différentes tonalités. [20]

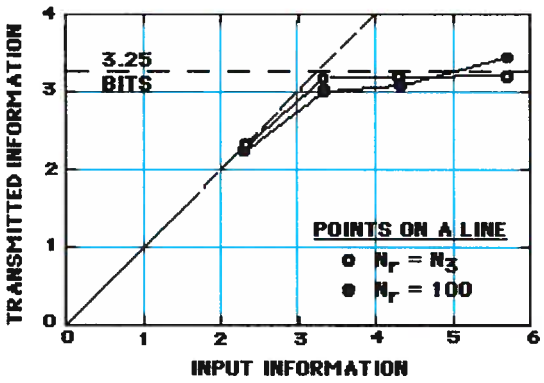


Figure 3 Courbe représentant le nombre de localités identifiées sans confusion [20].

Nous remarquons donc que la quantité d'informations transmises est largement plus grande lorsqu'il s'agit d'identifier des positions distinctes d'un marqueur sur une ligne. Celle-ci passe de 2.5 (6 stimuli) pour le premier cas (Figure 2) à 3.25 (10 stimuli) (Figure 3).

2.4 Le système visuel humain

Le système visuel humain est un système complexe qui a évolué dans le temps, il est essentiellement composé de deux sous-systèmes : le système de bas niveau appelé le système « où » et le système de haut niveau appelé le système « quoi ».

2.4.1 Le système visuel de bas niveau

Le système visuel de bas niveau a contribué à la survie de l'espèce, car il permet de détecter rapidement les mouvements qui pourraient être l'attaque d'un prédateur ou d'un quelconque rival. Il permet également de percevoir la profondeur, l'organisation spatiale qui est essentielle, entre autre, à l'orientation. Enfin, il permet de différencier les figures de l'arrière-plan [21]. En résumé, le système visuel de bas niveau constitue la partie « hardware » du système visuel humain qui est responsable de tous les aspects de traitement d'images. Toutefois, contrairement au système visuel de haut niveau il est insensible aux couleurs.

2.4.2 Le système visuel de haut niveau

Le système de haut niveau représente la composante cognitive ou « software » du système visuel humain. Il est fortement influencé par la mémoire, les connaissances, le contexte et les intentions [22]. Il permet essentiellement de reconnaître les objets et de percevoir les couleurs. Pour illustrer nos propos nous présentons deux figures. La première figure représente des cercles noirs entrecoupés de bandes blanches (voir Figure 4). En regardant brièvement cette figure pour la première fois, on ne peut pas percevoir l'existence de la forme d'un cube. Ceci n'est possible qu'après un certain laps de temps, grâce au fait que l'on connaisse à quoi ressemble un cube. Ceci démontre l'importance de la mémoire et des

connaissances. En outre, la figure 5 exhibe l'importance de la connaissance du contexte. En effet, après quelques moments passés à fixer l'image nous pouvons voir la scène du chien qui renifle le sol. En résumé, le système visuel de haut niveau nous permet de trouver l'information cachée même dans des scènes bruitées. Cependant, à l'encontre du système visuel de bas niveau il est lent et a une faible sensibilité au contraste.

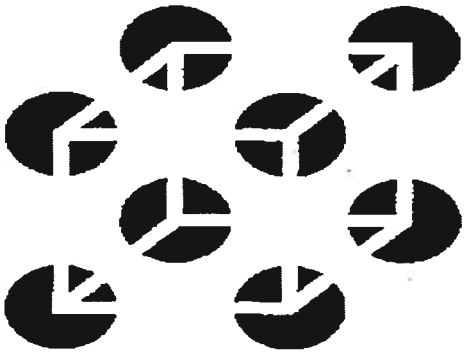


Figure 4 Du premier coup d'oeil on ne perçoit pas le cube[22].

Figure 5 **Exemple de scène non évidente à comprendre du premier coup : un chien qui renifle le sol [22].**

2.5 La visualisation et la stimulation

Dans la dernière section, nous avons exploré le système visuel humain, ses caractéristiques et ses composantes essentielles. Dans cette section nous nous intéressons aux facteurs qui peuvent nous aider à exploiter les atouts de ce dernier pour la production des applications de visualisation pertinentes et efficaces. C'est ainsi que nous nous intéressons, dans cette section, à l'utilisation de la stimulation multidimensionnelle et au contrôle de l'attention. L'utilisation des stimulations multidimensionnelles a une importance capitale car elle permet de créer des éléments de visualisation qui sont facilement différentiables et reconnaissables dépendamment de leurs attributs. Tandis que les techniques du contrôle de l'attention sont des techniques qui permettent surtout d'attirer et de capturer l'attention de l'utilisateur promptement pour lui signaler l'existence d'une information de grande importance.

2.5.1 Stimulation multidimensionnelle

Confronté à des stimuli unidimensionnels comme le changement d'un seul attribut visuel (la longueur, la couleur etc.), nous différencions seulement sept (plus ou moins deux) niveaux distincts sans confusion (section 2.3). Cependant, dans la vraie vie, nous pouvons facilement identifier avec exactitude les visages des gens qu'on connaît parmi des centaines de visages, un objet parmi des milliers d'autres, ou un mot parmi des milliers de mots. Ceci s'explique par le fait que plus le nombre d'attributs indépendants caractérisant un stimulus augmente, plus on dispose de niveaux de différenciations. Par exemple, les visages sont différents sur plusieurs aspects : la couleur de la peau, la couleur des yeux, la forme du visage, etc.

L'idée est donc d'utiliser une combinaison de différents attributs visuels pour pouvoir produire une visualisation facilement exploitable et utilisable. Toutefois, il faut faire attention à deux aspects importants : (1) l'aspect contrôle de l'attention qui permet de mettre l'emphase sur certains détails jugés importants. (2) l'aspect d'importance relative entre les différents attributs visuels, qui est très notable lors de la combinaison des différents attributs visuels.

2.5.2 Contrôle de l'attention

Confronté à un grand volume d'informations émanant de l'environnement, le système visuel effectue un filtrage en ignorant certains détails. Cette opération est communément appelée l'attention sélective. Il y a principalement deux mécanismes qui contrôlent le processus de sélection : le contrôle de la sélection dirigée par un but et le contrôle de la sélection influencée par un stimulus.

2.5.2.1 Contrôle de l'attention animée par un but (goal driven)

Dans ce cas, l'attention est influencée par les intentions, les motivations et les attentes de l'observateur. Considérons une personne qui est entrain de lire un livre dans un parc, elle va se concentrer sur le texte et ignorer les autres détails qui sont dans son champ de vision. Donc, le but de lire a fait en sorte que le système visuel filtre les autres détails qui ne sont pas pertinents pour accomplir la tâche de lecture. En d'autres termes, l'attraction de l'attention par le texte a été causée par les intentions et les motivations de la personne. D'ailleurs, dans les expériences sur l'étude du contrôle de l'attention, on demande souvent au sujet d'identifier ou de chercher, sur un écran, un objet ou un symbole ayant certains attributs. Cela veut dire qu'on crée dans ces conditions un contrôle de l'attention motivée par les attentes de l'observateur, l'attente que l'objet ayant l'attribut recherché apparaisse. Ce but pousse le sujet à ignorer tout autre stimulus. Toutefois, ce type de contrôle de

l'attention s'avère lent comparé au contrôle de l'attention par des stimuli que nous allons voir dans le prochain paragraphe.

2.5.2.2 Contrôle de l'attention par des stimuli (stimuli driven)

Les stimuli, surtout s'ils sont saillants, attirent rapidement l'attention indépendamment des buts et des croyances ou même de l'état d'esprit de la personne [27]. Un stimulus saillant est un stimulus qui a une caractéristique frappante. Ainsi, le mouvement, la discontinuité de la couleur et l'apparition brusque, sont autant d'exemples de stimuli saillants [25]. Par ailleurs, ces attributs appelés préattentifs sont détectés promptement par le système visuel de bas niveau d'une manière indépendante au nombre d'objets présentés. Cependant, la relation entre ce mécanisme d'attraction de l'attention et celui guidé par un but suscite plusieurs polémiques. Ainsi, dans la prochaine section nous allons explorer l'influence mutuelle des deux façons d'attirer l'attention.

2.5.2.3 Influence des deux façons d'attirer l'attention

D'une manière générale, il n'y a pas de consensus sur les facteurs qui permettent d'attirer l'attention. D'un côté, selon une partie de la communauté, le contrôle de l'attention est toujours influencé par le but. Cela veut dire qu'on ne détectera pas un stimulus préattentif si on n'a pas l'intention de le chercher. Ainsi, d'après ce que l'on nomme le modèle d'attraction contingenté, seulement les stimuli qui concordent avec des buts de recherche fixés à l'avance (goal driven) seront remarqués [28]. Tous les autres stimuli vont être ignorés. En effet, certaines expériences ont démontré que le système visuel humain peut nous faire défaut et laisser passer des détails importants qui resteront inaperçus. Un des exemples les plus frappants est celui de l'expérience conduite par Simons et Rensink [63]. Dans cette expérience, un professeur demande à ses étudiants de compter le nombre de passes que des joueurs de basket-ball vont se faire dans le film qu'il projette. Il y a deux équipes de joueurs les blancs et les noirs et on spécifie de compter les passes que les blancs se feront entre eux. En même temps que les joueurs se font des passes, il y a une personne

déguisée en gorille qui passe entre les joueurs et qui, à un moment donné, se met même directement devant la caméra. Après la projection, il s'est avéré que la moitié des étudiants, concentrés à compter les passes, n'ont même pas remarqué le personnage du gorille.

D'un autre côté, une partie de la communauté affirme que la notion d'attraction de l'attention est avant tout pilotée par les stimuli [30]. Cela peut être justifié par le fait qu'avant tout, les informations passent par le système visuel de bas niveau qui est sensible uniquement aux stimuli indépendamment de tout ce qui concerne le but, la connaissance, etc. C'est seulement après cette phase que les facteurs cognitifs interviennent. Une des expériences utilisées pour étayer ces propos est celle du double singleton. Il s'agit de demander à un sujet de chercher un singleton dans l'écran, mais au moment d'apparition du singleton on lui affiche un autre singleton qui n'a pas de rapport avec la tâche demandée. On mesure ainsi le temps de détection du premier singleton et on le compare à la situation où on affiche un seul singleton.

Finalement, une troisième vision réconcilie les deux premières. Cette vision préconise que la sélection guidée par les stimuli empiète sur la sélection guidée par un but et vice-versa dépendamment de certaines conditions [32]. Par ailleurs, un singleton de couleur interfère avec un singleton de forme. Cependant lorsqu'on ajoute d'autres formes –le singleton de forme n'est plus unique- le singleton de couleur n'interfère plus (voir figure 6). En fait, ceci s'explique par le fait qu'en cherchant un singleton de forme, le sujet se met en mode de recherche de singleton. Toutefois s'il remarque qu'il y a plusieurs formes il change de mode et se met en mode de recherche de propriété (feature search mode). Étant dans ce mode, la distraction par d'autres singletons incohérents par rapport à la tâche est automatiquement ignorée [33].

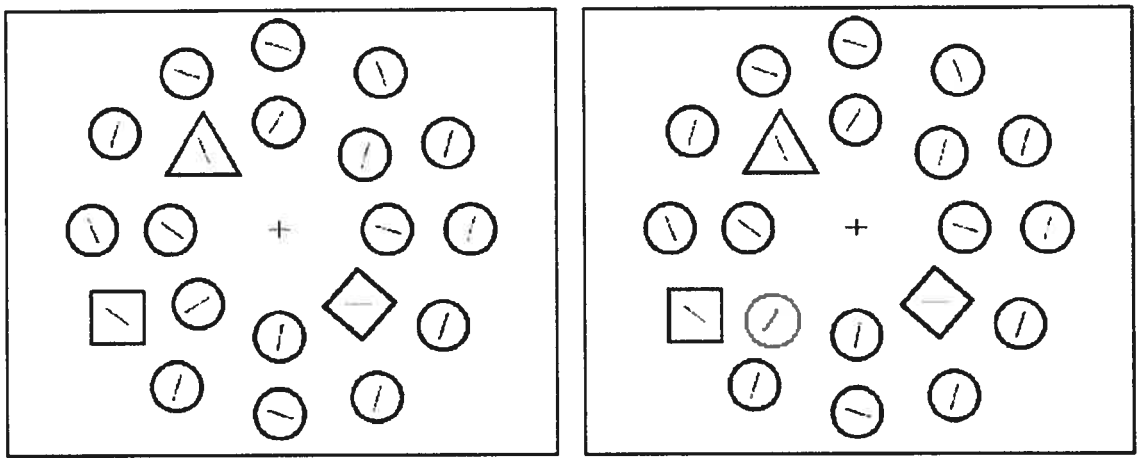


Figure 6 Expérience de recherche d'un singleton de forme, mélangé avec d'autres formes en l'absence de distraction (figure à gauche) et présence de distraction par la couleur (figure à droite)[32].

2.5.3 Importance relative entre les différents attributs visuels

Les attributs visuels saillants sont facilement détectables avec précision et rapidité dans un court laps de temps et indépendamment du nombre d'éléments exposés [34]. Utilisés adéquatement ils permettent de mettre l'emphase sur certains détails importants ou sur l'identification de frontières. Toutefois, il y a certaines hiérarchies entre ces différents attributs dont il faut tenir compte. Ainsi, comme le montre la figure 7 il est plus facile de détecter un singleton de couleur, soit le cercle rouge, parmi seulement des cercles bleus (b) que de détecter le même cercle dans un bain de carré de même couleur (c). Ce même principe s'applique pour l'identification des frontières figure 8. De plus, il est encore plus difficile de détecter ce même singleton lorsqu'on mélange les deux attributs visuels (forme et couleur) (Figure 7-f).

Figure 7 Importance relative entre les différents attributs visuels [34].

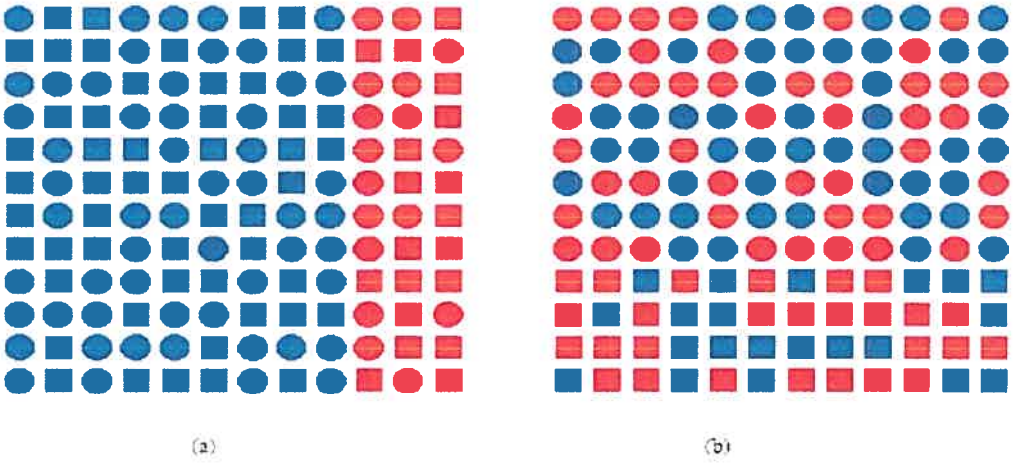


Figure 8 Différenciation des régions par la couleur et la forme[34].

2.6 Conclusion

Exploiter les capacités du système visuel humain tout en tenant compte de ses limites est le but de toute recherche dans le domaine de la visualisation. En effet, comprendre le fonctionnement du système visuel humain et des différentes méthodes de stimulation et de contrôle de l'attention est très important. Par ailleurs, ces connaissances sont des pré-requis essentiels pour comprendre le fonctionnement des systèmes de visualisation actuels dont nous présentons l'état de l'art dans le Chapitre 3 et pour le développement des systèmes de visualisation futurs que nous exposerons dans les chapitres 4, 5 et 6.

Chapitre 3

État de l'art: visualisation du logiciel

Ce chapitre offre un aperçu sur l'état de l'art de l'application de la visualisation dans le domaine du génie logiciel. Nous commençons par introduire la notion de visualisation de l'information qui est sans doute le fondement de l'application de la visualisation dans plusieurs domaines y compris celui du logiciel. Ensuite, nous nous intéressons à la classification de la visualisation du logiciel. Une telle classification nous permettra entre autre de bien définir notre système de visualisation, ainsi que de se positionner par rapport aux autres travaux fait dans ce domaine et qui sont présentés dans le reste des sections de ce chapitre.

3.1 Introduction

Malgré l'augmentation exponentielle de son volume et de sa variété, l'analyse et l'interprétation de l'information sont importantes voire même vitales. Par ailleurs, il y a plusieurs sources d'information dont les images captées par satellite, les données recueillies par une station de météo ou même les transactions faites à la bourse.

La recherche d'interprétation et du sens des données ont toujours été des besoins urgents que ce soit pour les scientifiques qui cherchent à comprendre un phénomène quelconque ou même pour les gens de marketing qui essayent de cibler leur public cible ou d'améliorer leur produit. La vision humaine, de par ses attributs comme la capacité d'interprétation rapide et sans effort d'un grand nombre d'information même dans un environnement bruité, constitue une candidate idéale pour combler ces besoins.

Le but de la visualisation de l'information est l'exploitation du système visuel humain pour extraire de l'information pertinente à partir des données. La visualisation est un moyen

essentiel pour l'identification des structures, des patterns, des anomalies et des relations. Celle-ci fournie une vue d'ensemble sur des données complexes et nous assiste dans l'identification des zones d'intérêt [19].

La visualisation du logiciel est la visualisation de tous les artefacts qui sont liés au logiciel durant toute sa durée de vie partant de l'analyse et la conception allant jusqu'à l'implémentation et les tests. D'ailleurs, comme le montre la figure 9 la visualisation du logiciel inclut celle de programmes et d'algorithmes. Toutefois, dans la plupart des cas, ces catégories s'empiètent, donc les frontières entre elles demeurent floues et discutables.

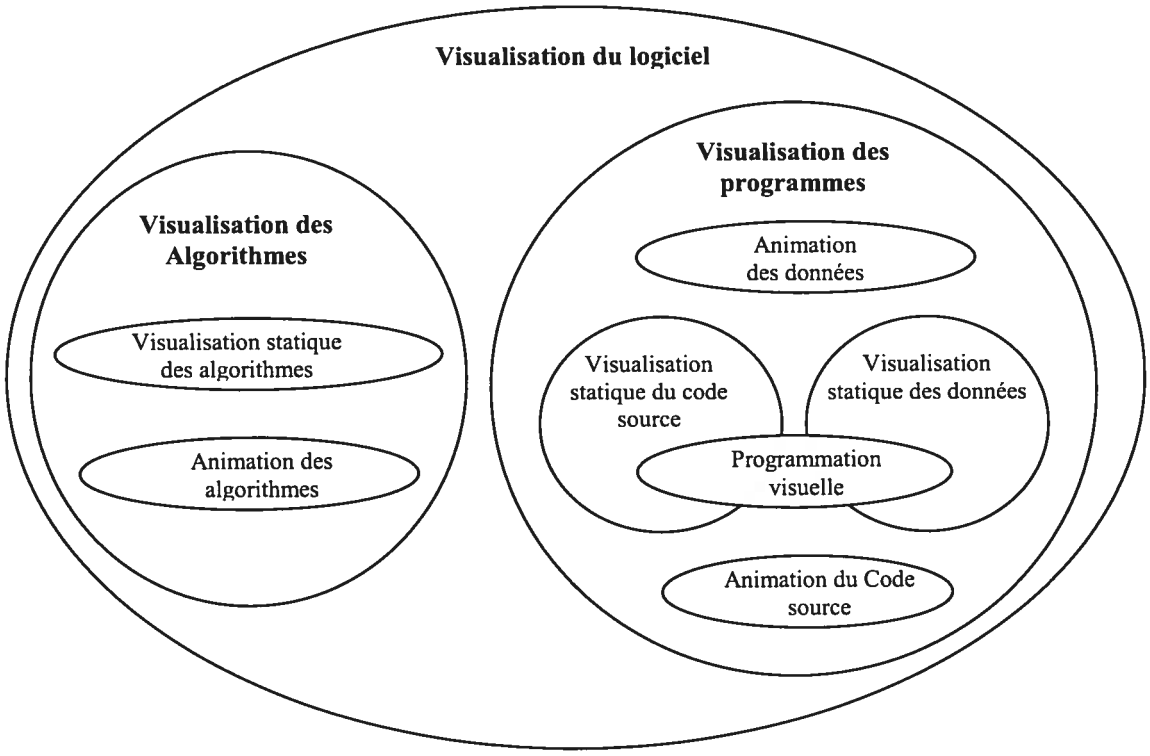


Figure 9 Composition de la visualisation de logiciel [38]

Globalement, les systèmes de visualisation du logiciel peuvent être répartis en deux catégories. D'un côté nous avons les systèmes de visualisation qui sont axés sur les informations extraites à partir de la structure statique du logiciel. D'un autre côté, nous avons les systèmes de visualisation qui s'intéressent surtout au comportement dynamique du logiciel. Par ailleurs, en se basant sur les différents critères et principes de

classifications plusieurs taxonomies ont été proposées pour les systèmes de visualisation du logiciel.

3.2 Classification des systèmes de visualisation du logiciel

Il y a principalement deux taxonomies différentes : la première est proposée par Roman et Cox et la deuxième est proposée par Price Baecker et Small. Roman et Cox [37] ont proposé un modèle de classification basé sur quatre critères (voir figure 10). (1) La portée : elle délimite les caractéristiques logicielles ciblées par la visualisation. Autrement dit : quels sont les intrants du système de visualisation. (2) L'abstraction : elle spécifie le niveau d'abstraction de la visualisation, par exemple la visualisation du code et des instructions est un bas niveau d'abstraction, alors que la visualisation d'architecture globale du programme est un haut niveau d'abstraction. (3) La méthode de spécification : elle concerne la manière de construire la visualisation. Est-ce que la correspondance entre les attributs visuels et les attributs du programme est figée (hardwired) ou peut être changée. (4)La technique utilisée pendant la correspondance qui dénote comment choisir les bons attributs visuels qui représentent le mieux certains attributs du programme.

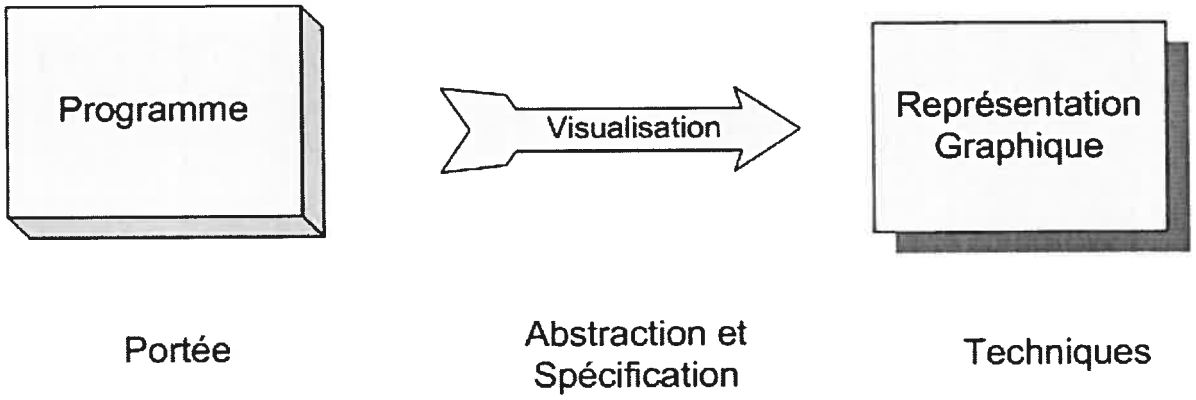


Figure 10 Modèle de taxonomie des visualisations de programmes, proposé par Roman et Cox.

Price, Baecker et Small [38] utilisent une taxonomie décrite par un arbre à plusieurs niveaux. Dans le premier niveau il y a six catégories principales qui sont extensibles, ce

qui permet de rajouter des sous catégories plus spécialisées. Ces catégories sont la portée, le contenu, la forme, la méthode, l'interaction et l'effectivité (voir figure 11).

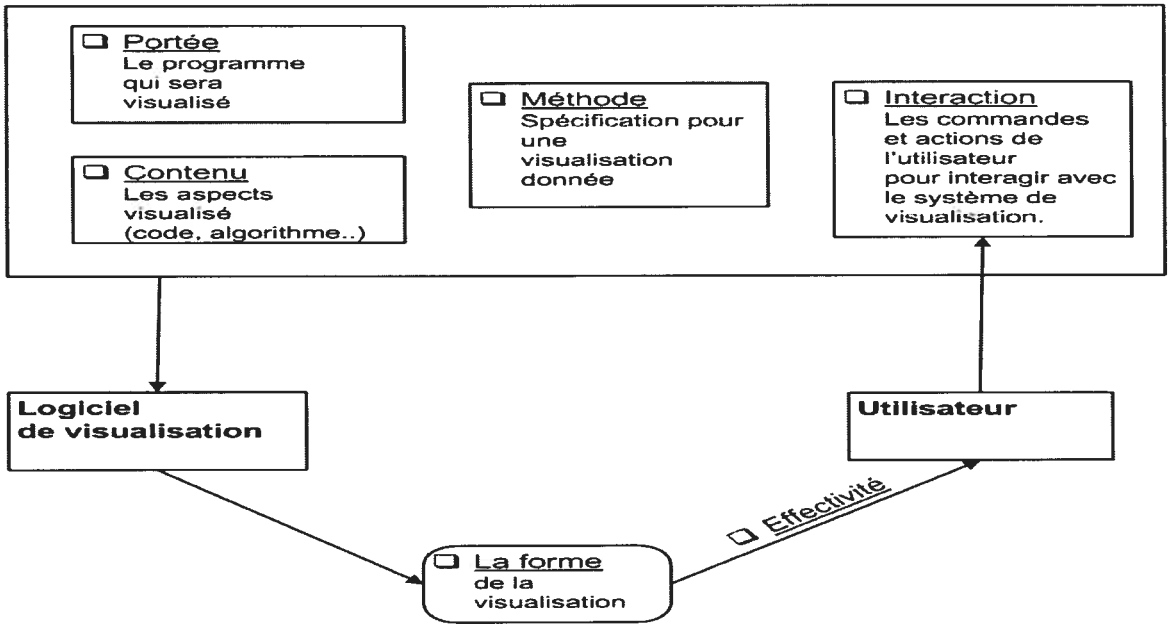


Figure 11 Modèle de taxonomie des programmes de visualisation proposée par Price, Baeker et Small.

(1) La portée définit la gamme de programmes que le système de visualisation prend comme entrée. (2) Le contenu discerne le sous-ensemble d'informations qu'on veut visualiser, ce peut être le code, les données, les algorithmes, etc. (3) La forme représente les caractéristiques du produit de la visualisation, est-ce que c'est de l'animation, du son ou du vocabulaire graphique. (4) La méthode : indique la façon par laquelle la visualisation est construite, si elle est générée automatiquement ? Quelles techniques ont été utilisées pour faire la correspondance entre les attributs du logiciel et les attributs visuels ? (5) L'interaction spécifie comment l'utilisateur interagit et contrôle le système. (6) L'effectivité est une mesure subjective qui permet d'apprécier jusqu'à quel point le système communique efficacement l'information à l'utilisateur. On remarque que la différence principale entre cette taxonomie et celle proposée par Roman et Cox est le fait de mettre l'accent sur deux aspects qui concernent l'utilisateur de la visualisation qui sont l'interaction et l'effectivité (voir figure 11).

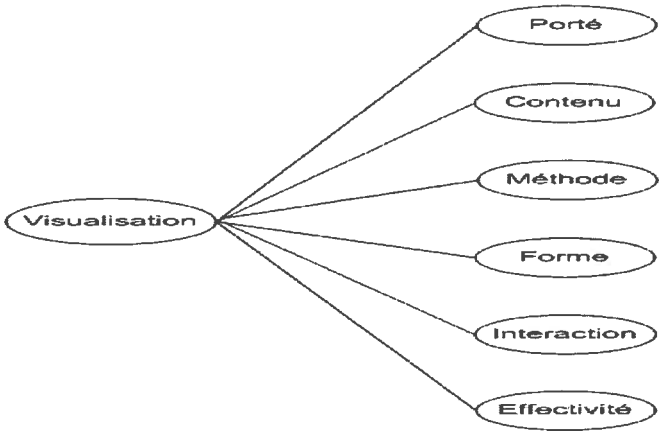


Figure 12 Modèle de taxonomie en forme d'arbre

3.3 La visualisation pendant le cycle de vie du logiciel

En s'inspirant du modèle de Price, Baecker et Small, nous proposons de regrouper les outils de visualisation du logiciel selon leur utilisation pendant chaque étape du cycle de vie du logiciel (voir figure 13). D'abord, pendant l'étape d'analyse de conception, nous avons recours à des outils permettant la description des éventuelles fonctionnalités du logiciel, de ses composantes et leurs relations et de l'interaction entre les différents acteurs et le système, etc. D'ailleurs, la majorité des outils de conception (Rational, Visio, etc.) utilisent les notations UML (Unified Modeling Language) qui sont composées principalement de différents diagrammes qui représentent à la fois la structure et le comportement du système (section 3.3.1). Ensuite pendant l'étape d'implémentation nous avons recours à plusieurs outils de visualisation du code (section 3.3.2.1) en tant que tel, ou des algorithmes (section 3.3.2.2). Finalement, durant l'étape de maintenance et de tests nous avons besoin d'outils permettant non seulement le rassemblement et la gestion des données d'exécution, mais aussi leurs visualisations de manière efficace qui facilite autant l'identification des sources de bug que la découverte de possibilités d'améliorations (section 3.3.3).

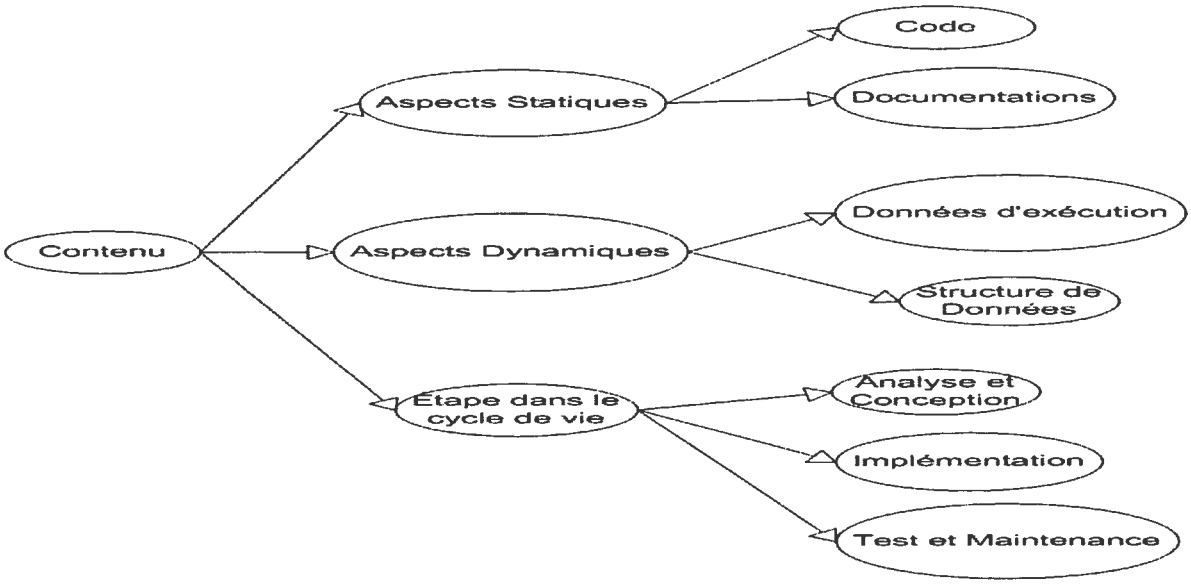


Figure 13 Catégorisation qui tient compte des étapes du cycle de vie du logiciel en plus des aspects dynamiques et statiques.

3.3.1 Visualisation pendant l'étape d'analyse et de conception

Dans la première phase du cycle de vie du logiciel et à un niveau d'abstraction élevé, on veut avoir une idée à propos de l'architecture générale du logiciel, des entités qui le composent et de leurs relations. De nos jours, l'UML est sans doute le langage de visualisation de choix à ce niveau. Il est ainsi utilisé par la majorité des développeurs pendant toute la durée de vie du logiciel, en partant de la conception et la spécification allant jusqu'à l'implémentation.

3.3.1.1 UML : aperçu

L'UML est composé de plusieurs types de diagrammes qu'on peut classer essentiellement en deux catégories [40]. La première catégorie englobe les diagrammes décrivant la structure statique du système. La deuxième est composée des diagrammes qui exposent le comportement de ce dernier (voir figure 14).

Ces diagrammes peuvent être utilisés durant toutes les étapes du cycle de vie du logiciel. Toutefois, leur utilisation est plus cruciale pendant la phase d'analyse et de conception. Durant cette étape, ils permettent en premier lieu de décrire et de spécifier les besoins (cas d'utilisation) qui sont sans doute la raison d'être du logiciel. En deuxième lieu, ces diagrammes donnent une vue d'ensemble du système en représentant toutes ses composantes (diagramme de classes) ainsi que l'interaction entre celles-ci (diagramme d'interaction). Ces représentations visuelles sont un moyen expressif et efficace de communication entre les équipes de conception et celles de développement, ce moyen minimise les mauvaises interprétations et les malentendus qui sont inévitables lors de l'utilisation du langage naturel seulement.

Toutefois, pour des logiciels de grande envergure les diagrammes UML deviennent plus grands et complexes. Cette complexité rend leur compréhension plus ardue. Par ailleurs, l'arrangement des différents constituants de ces derniers devient une tâche compliquée, qui requiert l'utilisation d'algorithmes de placement qui sont coûteux en temps de calcul et dont le résultat est souvent loin d'être esthétique ou lisible. Ce problème a poussé plusieurs chercheurs à proposer des extensions et des améliorations à ce modèle. C'est ce que nous découvrirons dans les deux prochaines sections (3.3.1.1, 3.3.1.2).

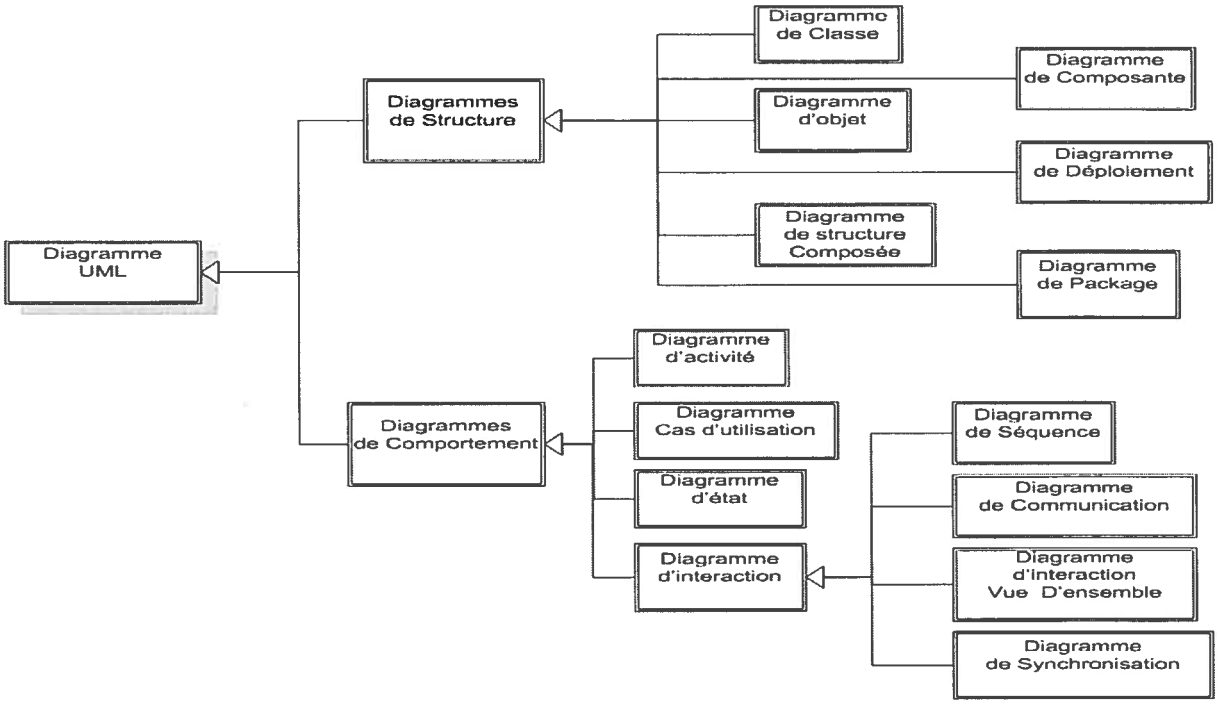


Figure 14 Classification des diagrammes UML [40]

3.3.1.2 UML : utilisation de la troisième dimension et de l'animation

Durant les deux dernières décennies, nous avons observé une grande évolution dans le domaine de l'infographie grâce à une baisse des prix des matériaux informatiques et un avènement des librairies graphiques puissantes et faciles à utiliser comme OpenGL et DirectX. Par conséquent, la troisième dimension, l'animation et les effets spéciaux sont maintenant couramment utilisés dans plusieurs domaines dont celui de la visualisation de l'information et du logiciel. D'ailleurs, plusieurs recherches dans le domaine de la visualisation du logiciel sont axées sur l'exploitation de ces nouveaux artefacts graphiques dans le but de passer des présentations unidimensionnelles et statiques aux animations et simulations tridimensionnelles et interactives.

Radfelder et Gogola [41] proposent une évolution de certains diagrammes UML en utilisant la troisième dimension, la combinaison de différentes vues et l'animation. En effet, ils ont d'abord suggéré l'utilisation de la troisième dimension pour résoudre les problèmes

d'arrangement des symboles, en mettant les plus importants au premier plan et les moins importants en arrière-plan (voir figure 15). Ensuite, ils ont suggéré une solution pour afficher deux aspects différents du logiciel dans une même vue. Ainsi, à titre d'exemple, ils ont permis la combinaison entre l'aspect statique comme le diagramme de classe avec un aspect dynamique comme le diagramme de collaboration (voir figure 16). De plus, ils ont profité de l'animation pour faire des transitions fluides entre différents diagrammes et simuler le passage des messages entre différents objets. Finalement, ils ont introduit la possibilité de navigation interactive, pour faciliter le changement entre les différentes vues et les différents niveaux de granularité.

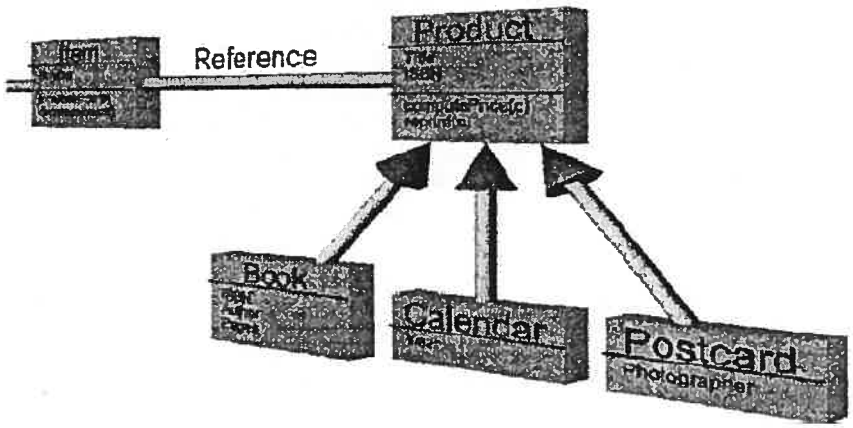


Figure 15 Une présentation de diagramme de classe en trois dimensions [41].

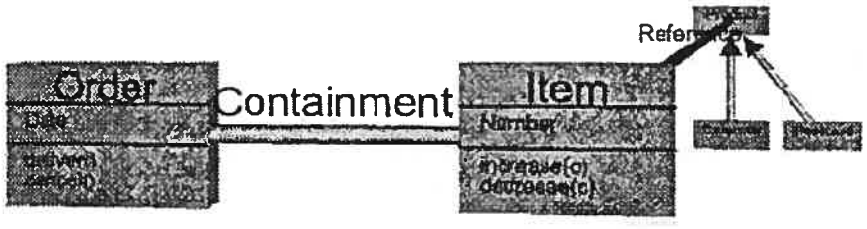


Figure 16 Mise en évidence de la zone d'intérêt, en mettant les symboles concernés en avant-plan et le reste en arrière-plan [41].

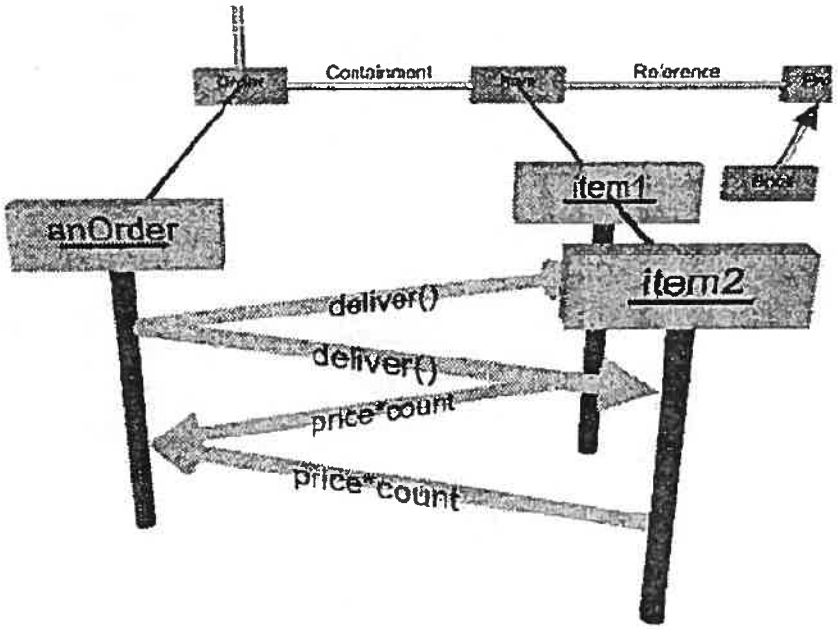


Figure 17 Une combinaison entre une vue statique (diagramme de classe) et une vue dynamique (diagramme de séquence).

3.3.1.3 UML : utilisation de la troisième dimension et des lois physiques

Outre l'emploi de la troisième dimension, Dwyer [42] explore l'utilisation de l'algorithme des forces dirigées (FDA). Cet algorithme a été adapté pour l'arrangement des symboles dans un diagramme de classes afin d'améliorer sa compréhension et sa lisibilité. Par ailleurs, cette méthode automatique débarrasse le développeur de la tâche monotone, longue et pénible de l'arrangement des symboles afin qu'il puisse se concentrer sur d'autres problèmes plus importants. En effet, Dwyer suggère que l'adoption d'un modèle en trois dimensions, utilisant les lois physiques, permet à l'utilisateur de comprendre les diagrammes d'une manière intuitive, puisque c'est un modèle régi par les lois physiques semblables à celles du monde où l'on vit.

L'algorithme de force dirigée, connu aussi sous le nom d'algorithme des ressorts, simule les forces d'attraction et de répulsion qui s'exercent entre les différents éléments d'un

graphe. La simulation s'arrête lorsque le système atteint un état d'équilibre qui minimise la somme de ses forces. Comme l'illustre la figure 18 chaque élément du graphe a une masse ou une charge (B) qui permet de calculer la force gravitationnelle, ou la direction de la force entre deux nœuds ce qui minimise le croisement des liens. D'ailleurs, par analogie, les mêmes types de forces qui sont utilisées entre les nœuds sont aussi utilisées pour positionner des agrégats de nœuds qui représentent les packages (voir figures 18 et 19).

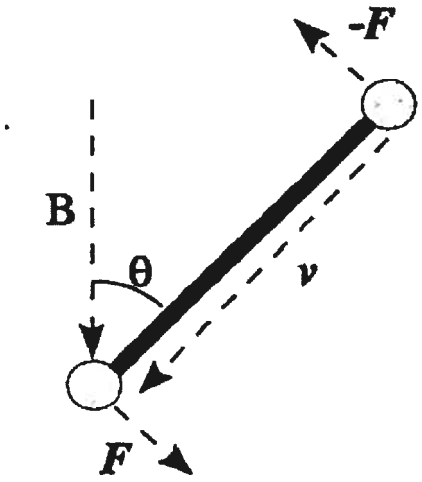


Figure 18 La représentation des forces qui s'exercent entre deux éléments. B représente la direction et la norme du champ magnétique, v est le vecteur de la force qui s'exerce entre les deux éléments n1 et n2, -F et F sont les forces résultantes sur n1 et n2.

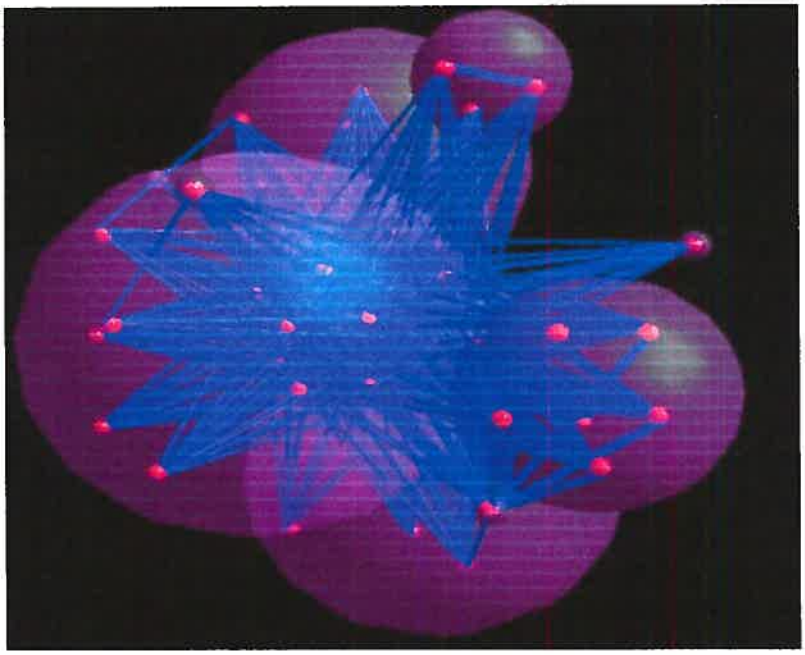


Figure 19 Figure produite par WILMA, les nœuds représentent les classes alors que les grandes sphères représentent les agrégats qui sont les packages

Ainsi, un programme nommé WILMA (figure 19) a été développé par Eckersley pour tester ces théories. WILMA est codé en java, en utilisant Java3D toolkit, son engin de graphe peut être utilisé non seulement pour représenter des diagrammes UML mais aussi pour n'importe quelle modélisation conceptuelle qui implique l'utilisation des graphes (voir figures 20 et 21). De plus, il offre un environnement qui supporte la navigation et l'interaction avec l'utilisateur.

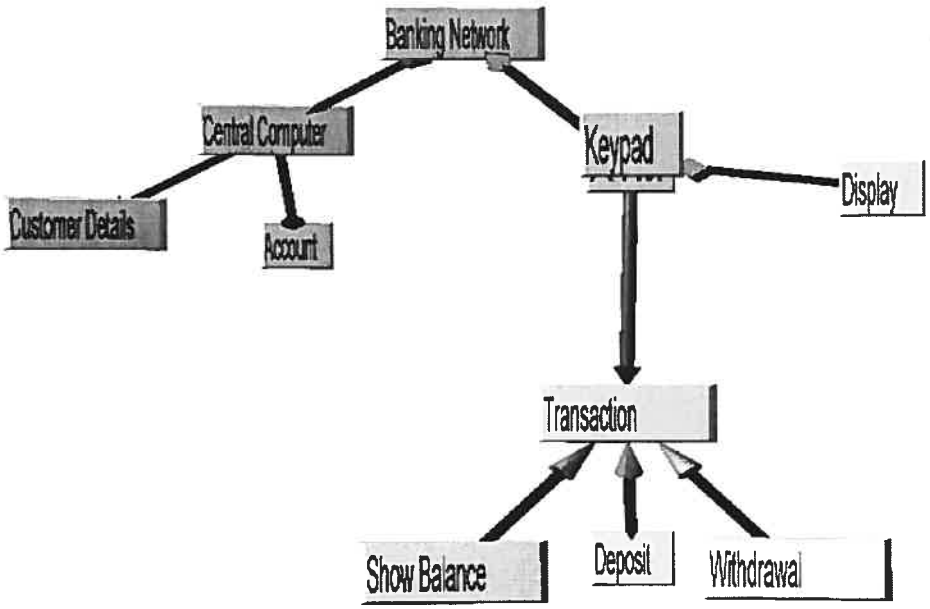


Figure 20 Une représentation d'un diagramme UML dans WILMA.

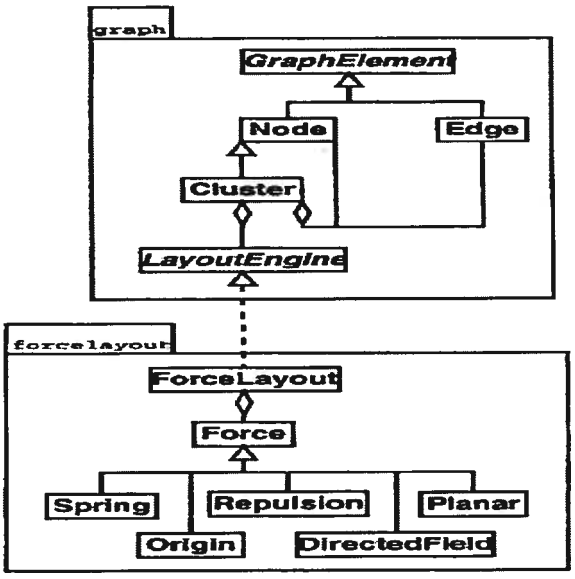


Figure 21 Figure qui représente un diagramme UML « classique ».

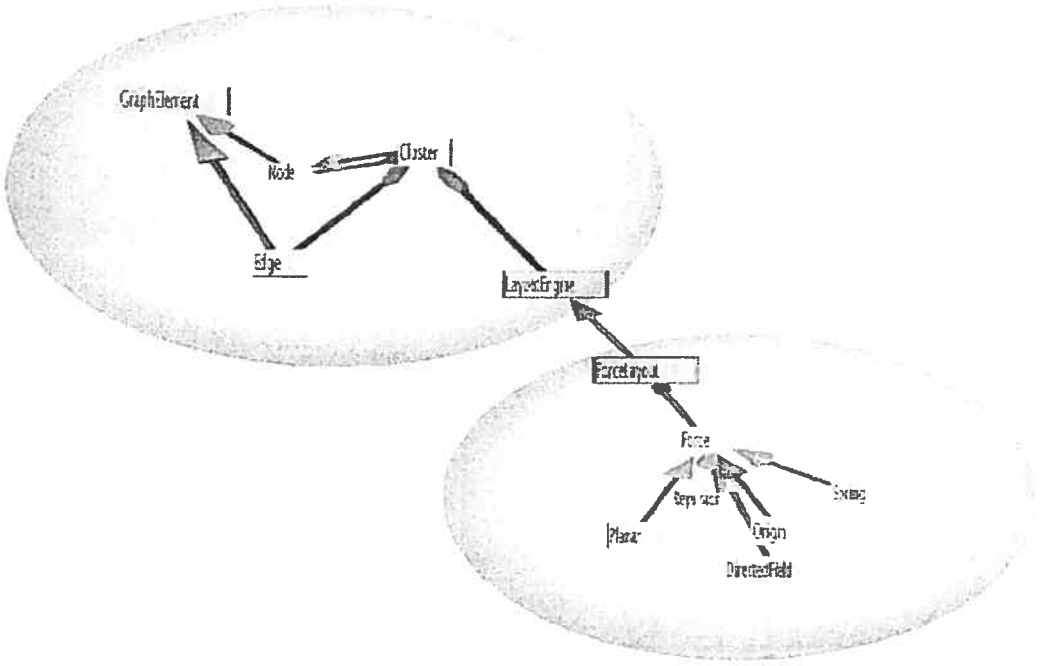


Figure 22 La présentation graphique produite par WILMA du digramme de classe présenté à la figure 21, la première sphère en haut à gauche représente le package graph. La deuxième sphère représente le package forcelayout.

Bien que ces outils de visualisation de structures générales offrent une vue globale du logiciel, de ces composantes, de leurs relations et de leurs interactions, il n'en demeure pas moins qu'on a besoin d'avoir une vue de moindre niveau d'abstraction et à un plus grand niveau de granularité. Par ailleurs, ce genre de vue nous permettra de mettre le logiciel sous le microscope pour observer les caractéristiques qui sont liées au code, aux algorithmes, aux chemins d'exécution et qui sont les produits de l'étape d'implémentation.

3.3.2 Visualisation pendant la période d'implémentation

Une fois la conception faite, on commence l'implémentation de programmes qui seront conformes aux spécifications prédéfinies. Ces programmes augmentent de complexité et de volume au fur et à mesure qu'on avance dans le processus de développement. Par conséquent, durant cette phase, on a besoin de moyens facilitant la perception d'une vue d'ensemble du programme et de ses fichiers ; ou encore, des moyens favorisant la compréhension de son comportement et de ses algorithmes. La visualisation de programmes s'adresse entre autres à ces besoins.

La visualisation du programme est l'utilisation de plusieurs techniques de visualisation pour améliorer la compréhension des programmes. En revanche, la programmation visuelle est l'utilisation de techniques visuelles pour construire des programmes [38]. La visualisation de programmes a pour but de montrer ses aspects cachés afin de rendre plus évident ses attributs et caractéristiques, qui auraient été difficiles à évaluer autrement. La visualisation de programmes inclut la visualisation d'algorithmes, mais encore plus, elle couvre d'autres aspects relatifs aux caractéristiques du programme qui sont entre autres : son code, ses points de contrôle et son comportement.

3.3.2.1 Visualisation du code source

À travers l'analyse statique du code source, nous pouvons extraire beaucoup d'informations concernant le logiciel, autant sur sa structure générale que sur la complexité d'une ligne de code. Nous présentons dans ce qui suit quelques exemples de système de visualisation du code.

Seesoft est une métaphore simple et intuitive selon laquelle chaque ligne de code est représentée par un carré dont la couleur correspond à une caractéristique statistique [5], qui peut être par exemple le nombre d'instructions, ou même le niveau d'imbrications (voir figure 23). Cette correspondance directe entre le code source et la visualisation permet une

navigation naturelle entre les deux représentations, ce qui consolide la confiance de l'utilisateur du système de visualisation. De plus, elle permet de représenter un grand nombre de fichiers de grande taille dans une seule localité spatiale. Par ailleurs, l'observation d'une telle représentation donne une idée globale sur le système, sur son ampleur et sa complexité. Grâce à ces qualités ce modèle a été repris par d'autres chercheurs pour l'améliorer.

Le modèle SV3D est la nouvelle version de Seesoft : il a été élaboré par Marcus et Al [6]. D'abord, son atout principal est l'ajout d'une troisième dimension qui a permis d'augmenter son expressivité. Par ailleurs, comme le montre la figure 24 ce modèle supporte des dimensions additionnelles qui sont : la forme du poly-cylindre, sa hauteur dans l'axe des Z+, sa profondeur dans l'axe des Z. Ensuite, ils ont introduit une flexibilité au niveau de la correspondance entre les attributs visuels et les attributs du logiciel. Ainsi le système suggère une correspondance par défaut qui peut être changée à la guise de l'utilisateur. Enfin, la possibilité de *zoom in* et *zoom out* et de navigation évite la surcharge de la vue et améliore l'effectivité et l'expressivité du système.
Bien que ces présentations valorisent les analyses à un niveau de granularité fin, qui peut aller jusqu'à la présentation des instructions du programme, elle ne permet pas de découvrir d'autres aspects importants du logiciel comme son comportement et ses algorithmes. C'est ce que nous découvrirons dans la prochaine section.

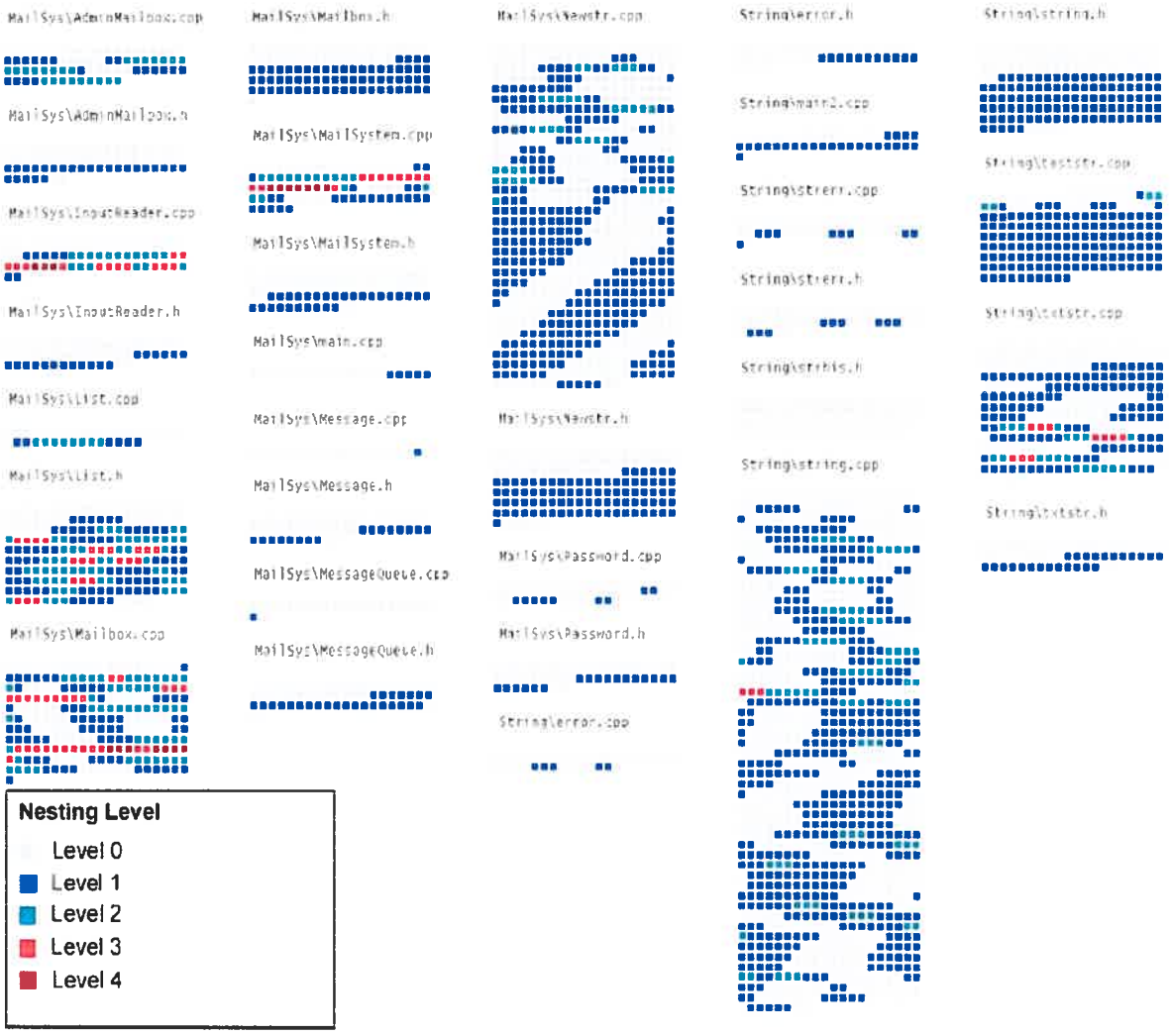


Figure 23 Au dessous de chaque nom de fichier, les carrés représentent ses lignes de code. La couleur de ces derniers représente le niveau d'imbrication.

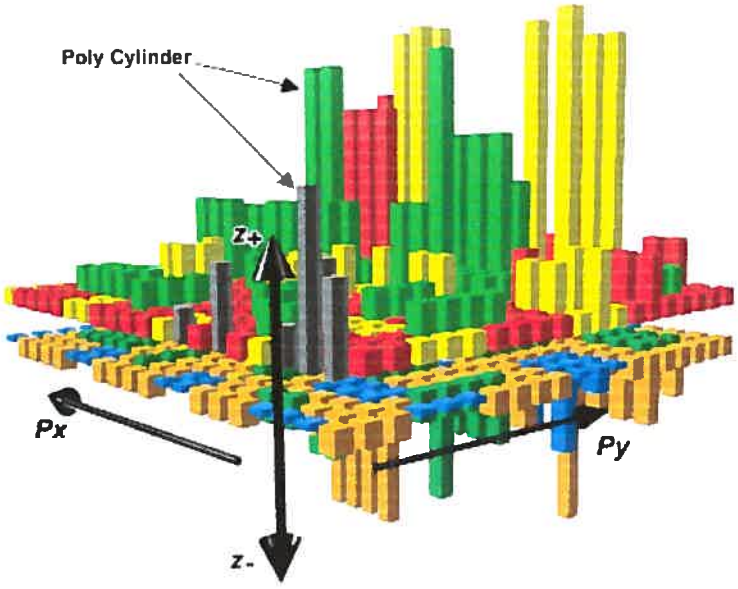


Figure 24 Les éléments de visualisation utilisés par SV3D, qui sont la forme du poly-cylindre, sa profondeur Z-, sa hauteur Z+ , sa couleur et sa position.

3.3.2.2 Visualisation d'algorithmes

La visualisation d'algorithmes est consacrée à l'exposition de leurs propriétés caractéristiques en utilisant les moyens graphiques adéquats. Souvent elle se présente sous forme d'animations réalisées par l'insertion d'appels à des routines graphiques dans le code source du programme.

La visualisation d'algorithmes est la catégorie la plus utilisée dans le domaine du logiciel. Elle représente un outil puissant qui aide les enseignants à expliquer certains algorithmes complexes qui sont difficiles à expliquer par des textes ou oralement. De plus, elle offre un grand support aux apprenants qui peuvent observer l'animation de l'algorithme pendant l'exécution simultanée du programme. Finalement, elle permet la comparaison entre différents algorithmes, et entre les variétés d'implémentations d'un même algorithme. Dès

lors, la visualisation est un moyen puissant de compréhension, d'évaluation et d'amélioration des algorithmes [2].

On peut discerner globalement deux types d'outils de visualisation d'algorithme, (1) les outils de visualisation passive et (2) les outils de visualisation active. L'apprenant utilisant le premier type de visualisation joue le rôle d'un spectateur, il regarde la visualisation d'algorithmes faite par un expert (l'instructeur) sans aucune possibilité d'intervention. Le surplus de compréhension gagné grâce à ces outils s'avère marginal. En revanche, le deuxième type d'outils donne à l'apprenant un rôle actif. Ainsi, il peut intervenir avant, pendant et après la visualisation [35]. Avant, en changeant ou en spécifiant les entrants, ou encore mieux, en lui permettant de faire une visualisation qui exprime sa propre compréhension. Pendant, en ayant la possibilité d'avancer ou de reculer l'animation, ce qui lui permet de comprendre à son propre rythme. Et finalement après, en ayant la possibilité de partager et de comparer ses visualisations avec celles d'autres étudiants [4].

Enfin, il faut noter qu'en tenant compte de l'ampleur des applications distribuées, il y a de plus en plus d'études qui se font dans le domaine de la visualisation des algorithmes distribués. Ces études ont donné naissance à des techniques de visualisation qui offrent différents avantages. Premièrement, un support à la compréhension de la communication entre différentes composantes du logiciel distribué, deuxièmement, un support à la représentation visuelle à la fois de leurs comportements et de leurs inter-collaborations [1].

3.3.3 Visualisation pendant l'étape de test et maintenance

La maintenance est une étape coûteuse qui consomme en moyenne 50% du budget du projet. Cela est souvent causé par des délais restreints ou un budget limité. Ainsi, certains logiciels sont livrés avec des fonctionnalités manquantes, des défauts majeurs ou même sans être testé extensivement dans différents environnements.

Il y a essentiellement trois types de maintenance [43]. La maintenance corrective dont le but est essentiellement de corriger le logiciel de manière à ce qu'il satisfasse les besoins et

les spécifications du départ. La maintenance adaptative qui concerne tout changement nécessaire pour s'adapter aux aléas de l'environnement du logiciel comme le changement du hardware, des compilateurs et du système d'exploitation. Finalement, la maintenance perfective dont l'objectif est l'amélioration ou l'ajout de nouvelles fonctionnalités qui ne font pas partie de la spécification d'origine.

Il y a beaucoup de travaux qui portent sur la maintenance et la visualisation. Toutefois, dans cette section nous nous intéressons seulement à la maintenance corrective. Nous citerons dans cette optique l'exemple de visualisation des données d'exécution. Nous reviendrons sur les autres aspects de la maintenance, qui sont en relation avec la qualité, un peu plus loin dans la section (3.3.4).

3.3.3.1 Visualisation des données d'exécutions des logiciels déployés

Une fois déployé, il est difficile d'avoir accès aux données d'exécution du logiciel dont la taille et le flot sont considérables. Il fallait donc, primo, trouver un moyen de gérer la collecte et la persistance de ces données. Secundo, avoir des techniques efficaces permettant l'extraction de l'information utile et pertinente à partir de ces données brutes.

Orso, Jones et Harrold [65] présentent l'environnement Gamma, technologie qui répond aux besoins cités ci-haut. En fait, il offre trois catégories de vues ayant trois niveaux différents de granularité (figure 29). À un niveau fin de granularité Gamma utilise la coloration des lignes de code pour identifier les instructions avec différentes couleurs (voir figure 25). Ensuite, à un autre niveau plus élevé de granularité le paradigme Seesoft (3.3.2.1) est utilisé pour la visualisation des lignes de code, cette technique est sans doute efficace car elle a déjà fait ses preuves. Toutefois, puisque la vision du code est loin d'être suffisante, Gamma offre aussi une vue d'ensemble du système. Celle-ci est élaborée en combinant la technique du « TreeMap view » développée par Schneiderman [44] ainsi qu'un mécanisme de coloration.

La visualisation en utilisant les « TreeMap views » est une approche de remplissage d'espace en deux dimensions dont le résultat est une structure d'arbre dans laquel chaque nœud est un rectangle dont la surface est proportionnelle à un attribut quelconque de ce nœud. Ce paradigme peut être utilisé pour présenter, par exemple, une structure des packages d'un système ainsi que leurs classes (voir figures 25 et 26).

Pour le mécanisme de coloration, Gamma utilise deux dimensions : le contraste et la luminosité. Le contraste, par analogie avec les lumières de circulation, utilise un spectre continu de couleurs. Ce spectre part du vert indiquant que les instructions de cette couleur ne requièrent pas d'attention particulière, allant au rouge qui indique totalement le contraire. Ensuite, on utilise la luminosité qui a une valeur allant de 0 jusqu'à 1 dont l'interprétation diffère selon le niveau de granularité (voir figure 28).

```
...
finallyMethod.setName(
    handlers.getFinallyNameForCFGStartOffset( finallyStartOffsets[i] ),
if ( numFinallyBlocks != 0 ) {
    finallyMethod.setType(PrimitiveValueOrPrimitiveVOID()),
    finallyMethod.setContainingType(parentMethod.getContainingType()),
}
finallyMethod.getContainingType().getProgram().addSymbol( finallyMethod ),
finallyMethod.setDescriptor( new String("()V") ),
finallyMethod.setSignature( parentMethod.
...
```

Figure 25 Vue au niveau des instructions

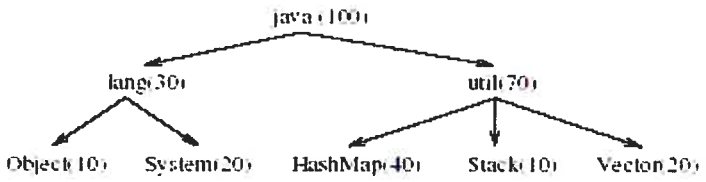


Figure 26 Vue d'arbre traditionnelle.

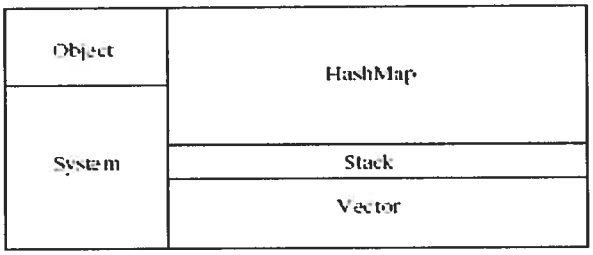


Figure 27 Structure d'arbre produite par l'algorithme de remplissage

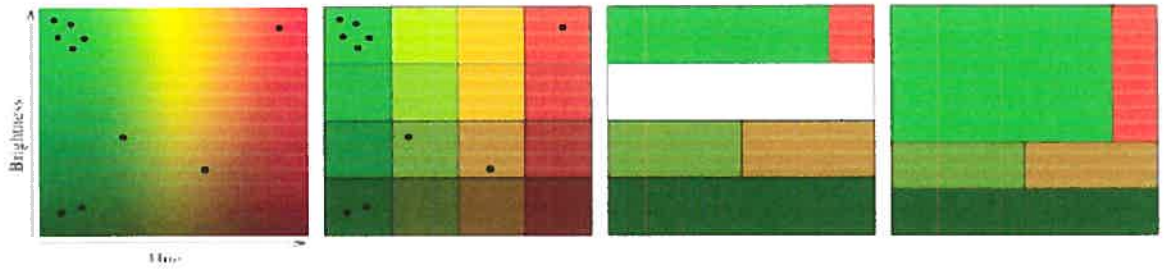


Figure 28 Combinaison de l'utilisation de la technique de partition d'espace et celle de coloration.

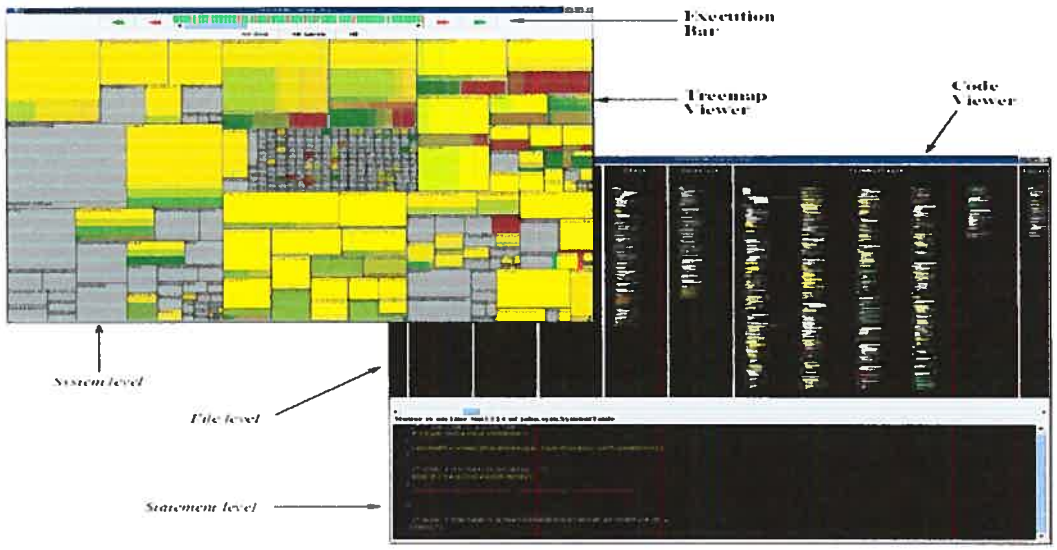


Figure 29 Une présentation des différentes vues générées par Gamma. On observe ainsi différents niveaux de granularité et la possibilité de navigation entre les différentes vues.

3.3.4 Visualisation des aspects quantitatifs et qualitatifs du logiciel

3.3.4.1 Métrique et qualité du logiciel

À chaque révision du logiciel on a besoin de savoir jusqu'à quel point ce dernier répond aux besoins fixés d'avance, c'est ce qu'on appelle la qualité de conformité aux besoins. Avant tout c'est la cause même d'existence du logiciel. A priori, cela peut être accompli par l'exécution et le test des fonctionnalités du logiciel une à une afin de calculer le taux de défaut. Toutefois, une telle approche est certainement irréaliste pour des logiciels de grande complexité et d'envergure. De plus, elle donnera naissance, si elle est appliquée, à des logiciels difficilement maintenables. Il faut donc avoir un moyen efficace de prédiction de la qualité du logiciel et du processus de sa production.

La qualité du logiciel est un concept multidimensionnel. Chaque dimension est une caractéristique intrinsèque du logiciel. Combinée avec des modèles de qualité, les métriques sont un moyen de mesure et d'appréciation de chacune de ces caractéristiques (voir figure 30). Par ailleurs, les métriques du logiciel peuvent être classifiées en trois catégories : métrique du produit, métrique du processus, métrique du projet. Les métriques du produit décrivent les caractéristiques du produit comme la taille, la complexité et les performances. Les métriques du processus concernent le développement du logiciel et sa maintenance, par exemple l'efficacité de correction des erreurs et la durée de temps de la détection de l'erreur au processus de correction. Les métriques de projet décrivent les caractéristiques du projet et de son exécution. Comme par exemple le nombre de développeurs, les coûts de développement, la productivité [46], etc. Par ailleurs, avec le surgissement du paradigme de la programmation orientée objet des nouvelles métriques plus spécifiques ont vu le jour.

Depuis l'apparition du paradigme orienté objet, le nombre de langages de programmation orientée objet (OOP) n'a pas cessé d'augmenter. Maintenant on parle de design orienté objet (OOD), d'analyse orientée objet, ainsi que de modélisation orientée objet. Ce paradigme gagne de plus en plus de terrain. Ses pilons de base sont les concepts de classe, d'objet, de méthode, de variable d'instance et d'héritage. Ainsi, les métriques objet ont pour rôle de décrire ces concepts en mettant, entre autre, l'emphase sur leurs caractéristiques, leur évolution et les relations entre eux, pendant toute la durée de vie du logiciel. Toutefois le calcul des métriques génère beaucoup de données, surtout dans des systèmes complexes ou de grande taille. Ceci rend difficile leur analyse et interprétation.

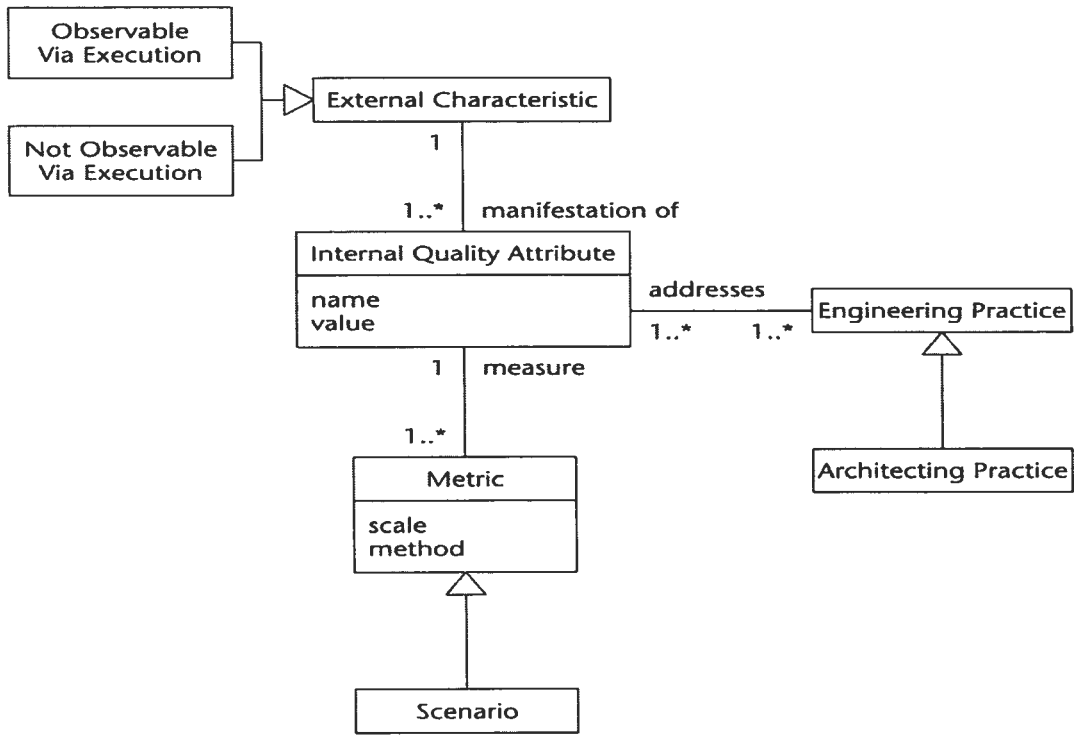


Figure 30 Un Méta Modèle de qualité [45]

3.3.4.2 Visualisation et métrique

La combinaison des métriques avec les techniques de visualisation offre une assistance à la compréhension du logiciel, l'évaluation de sa qualité et la détection de ses anomalies. Une des applications de visualisation de métrique les plus connues est le diagramme de Kiviat [47]. Le diagramme de Kiviat, communément appelé « *star graph* » ou « *spider graph* », permet la représentation de plusieurs métriques ayant différents intervalles de variation dans le même graphique. Comme l'illustre la figure 31 chaque axe représente une métrique. Sur le petit cercle bleu on trouve les valeurs minimales et sur le grand cercle rouge on trouve les valeurs maximales. Prenons un exemple : pour chaque classe on prend les valeurs de certaines métriques et on les marque sur les axes respectifs, par la suite on relie ces points formant un polygone. Une telle représentation permet de faire l'analyse des attributs de la classe par rapport à leurs valeurs minimales et maximales qui pourraient être des valeurs de limite d'acceptation. De plus, à un niveau plus haut de granularité, cette illustration permet de détecter les différents patterns qui pourraient suggérer l'existence d'anomalies.

Au niveau de la recherche dans le domaine de visualisation de logiciel, peu de chercheurs s'intéressent à la visualisation des métriques. D'ailleurs, d'après l'étude conduite par Rainer Koschke [48] auprès de 82 chercheurs en visualisation de logiciel dans les domaines respectifs : maintenance de logiciel, reverse engineering et re-engineering, seulement 25 d'entre eux utilisent la visualisation des métriques, contre 52 qui utilisent la visualisation pour le reverse engineering (voir Figure 31 Diagramme de Kiviat).

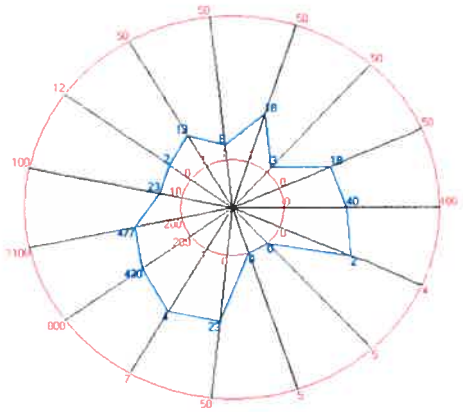


Figure 31 Diagramme de Kiviat

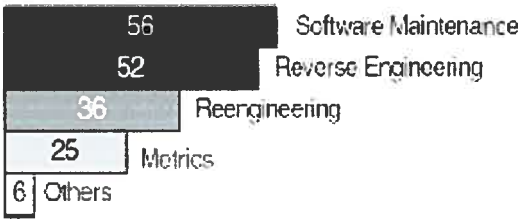


Figure 32 Histogramme présentant le pourcentage d'études et de projets de visualisation dans le domaine du génie logiciel qui touchent à chacune des branches de maintenance, reverse engineering, reengineering et les métriques.

3.3.4.3 Visualisation et reverse engineering

À cause de l'évolution technologique, l'apparition de nouveau paradigme comme la programmation orientée objet et les changements de réglementations, il y a toujours un besoin de modification et de mise à niveau des logiciels légués. D'ailleurs, partant du principe de réutilisation on voudrait récupérer certaines composantes des logiciels légués. Cependant, à cause du manque de documentations et de nombreux changements effectués par différentes équipes de développement à travers le temps, la réutilisation et la

maintenance de tels systèmes s'avèrent ardues. C'est pour cette raison qu'on a recours au reverse engineering.

Le processus de reverse engineering est composé essentiellement de cinq tâches comme le montre la figure 33. (1) L'extraction des artéfacts de bas niveau à partir du code source, (2) l'agrégation de ces artéfacts dans un modèle hiérarchique, (3) l'évaluation du modèle et l'exécution à nouveau de l'agrégation si nécessaire, (4) la sélection d'une sous hiérarchie si le système est grand et (5) la visualisation de ces artéfacts et leurs relations en produisant des représentations graphiques [49].

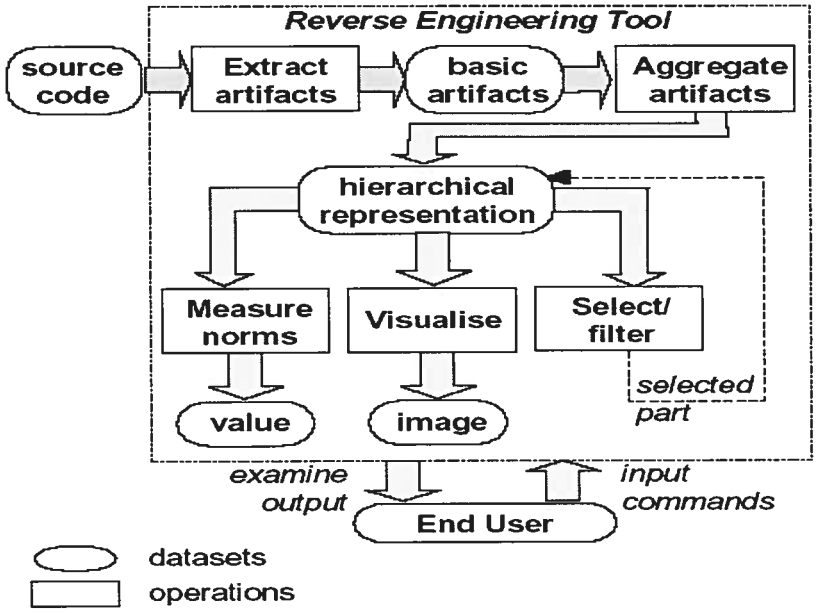


Figure 33 Reverse engineering pipeline [49].

Telea, Maccari et Riva proposent un ensemble d'outils [49] de prototypage de visualisation pour le domaine dureverse engineering. Ces outils permettent la génération de graphes (voir figure 34) dont les nœuds représentent les classes ou des packages et les liens présentent les relations entre ces derniers. D'ailleurs, chaque nœud possède un ensemble de caractéristiques regroupées dans une table de hachage sous format clef valeurs. Ces caractéristiques représentent des métriques qui peuvent être changées dynamiquement. On peut aussi éditer la structure du graphe en ajoutant ou en enlevant un ou plusieurs nœuds et

liens. Finalement, ces outils offrent une assistance à l'exploration de la visualisation, en permettant la navigation par sélection de zone d'intérêt (voir figure 34)

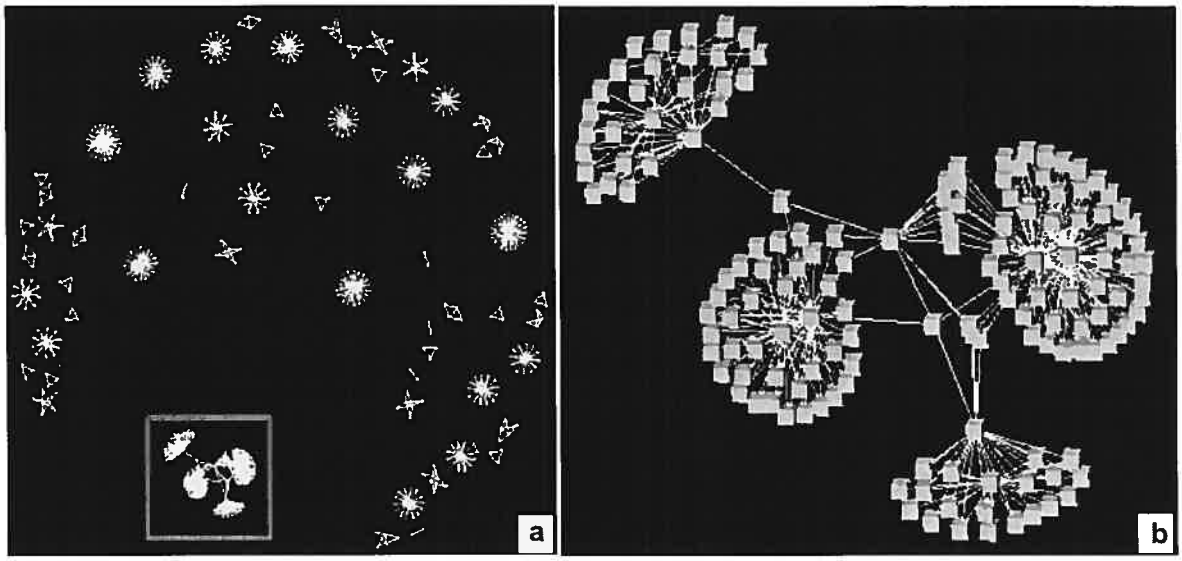


Figure 34 (a) Une vue d'ensemble de tout le système. (b) Une vue plus rapprochée de la zone sélectionnée en (a).

Lanza et Ducasse [50] mettent l'emphase non seulement sur la visualisation de classes et de leurs caractéristiques qui constituent le fondement de tout système objet, mais aussi sur l'aspect évolutif du système.

D'abord, à un niveau fin de granularité, on montre une représentation qui aide à la compréhension de l'implémentation concrète de la classe, en visualisant ses interfaces, ses méthodes les plus importantes et ses attributs (voir figure 35). De plus, une correspondance est établie entre certaines métriques et la longueur et largeur de chaque rectangle représentant une méthode ou un attribut. La couleur est aussi utilisée pour véhiculer de l'information supplémentaire. Comme illustré à la figure 35, nous pouvons remarquer que les attributs sont en bleu, les méthodes abstraites sont en cyan, les méthodes

sur définie sont en marron, etc. Ceci permet de détecter d'éventuelles régularités ou styles de programmation.

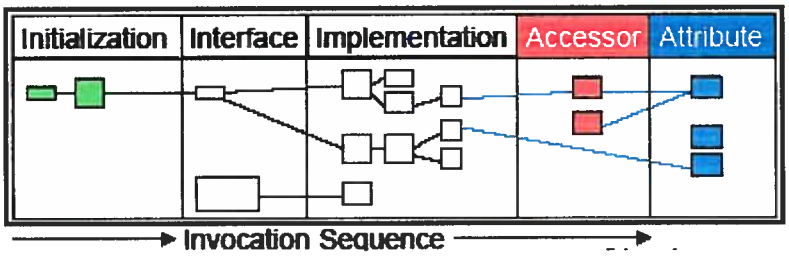


Figure 35 Représentation d'une classe. Ces caractéristiques sont affichées selon l'ordre d'invocation de gauche à droite, dans les rectangles respectifs.

Quant à l'aspect évolutif, on a utilisé une structure matricielle [51]. Chaque colonne de la matrice représente une version du logiciel, alors que chaque ligne représente les différentes versions de chaque classe (voir figure 38). Une telle structure permet d'observer l'évolution à deux niveaux. Premièrement, au niveau de la classe, elle est représentée par un rectangle dont la longueur et la largeur correspondent respectivement au nombre de ses variables et au nombre de ses méthodes. Ainsi, sur une même ligne on peut remarquer les modifications éventuelles de la classe, qui peut grandir, rétrécir ou rester stable. Pour rendre plus explicite de telles variations on a utilisé les couleurs gris pâle pour les classes stagnantes, ou diminuant de taille, et noire pour les classes qui augmentent de taille. Ainsi, on peut dégager certaines régularités. À titre d'exemple, les classes pulsar (voir figure 39) qui changent de taille à chaque version indiquant qu'elles sont des classes importantes du système puisqu'elles ont été modifiées à chaque version. Les classes supernova (voir figure 40) soudainement augmentent énormément de taille. Ceci peut être expliqué par un grand changement du système qui fait migrer beaucoup de méthodes vers cette classe, ou encore un prototype de classe qui vient d'être développé. Deuxièmement, au niveau du système, une telle présentation donne une vue globale du système et de son évolution à travers le temps. Par ailleurs, le nombre de lignes dans la matrice donne une appréciation de la taille du système. De plus, l'ajout et l'enlèvement des classes peut être remarqué facilement par l'ajout de nouvelles lignes (correspondant aux nouvelles classes) ou la

disparitions de certains rectangles. Les phases de stagnation ou de croissance peuvent ainsi être observées clairement (voir figure 41).

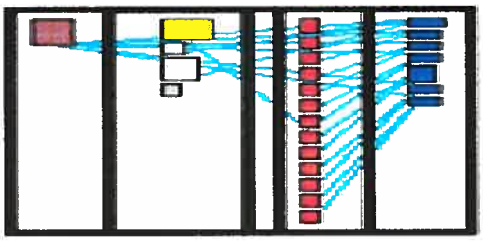


Figure 36 Un exemple de représentation d'une classe.

Figure 37 Un rectangle pouvant représenter une méthode ou un attribut.

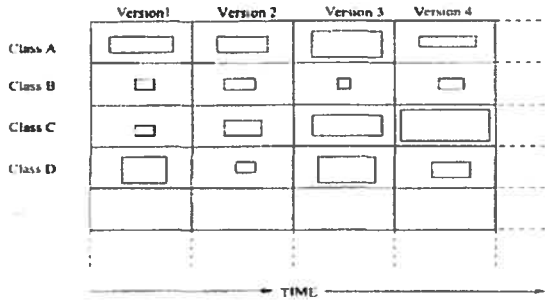


Figure 38 Matrice d'évolution du logiciel. Chaque ligne représente les différentes versions d'une même classe.

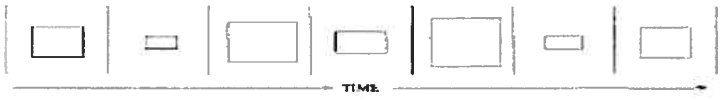


Figure 39 Une classe pulsar.

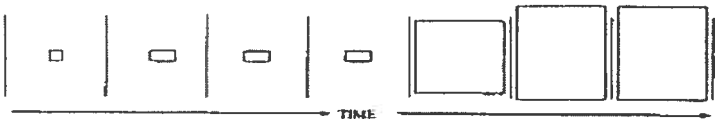


Figure 40 Une classe supernova.

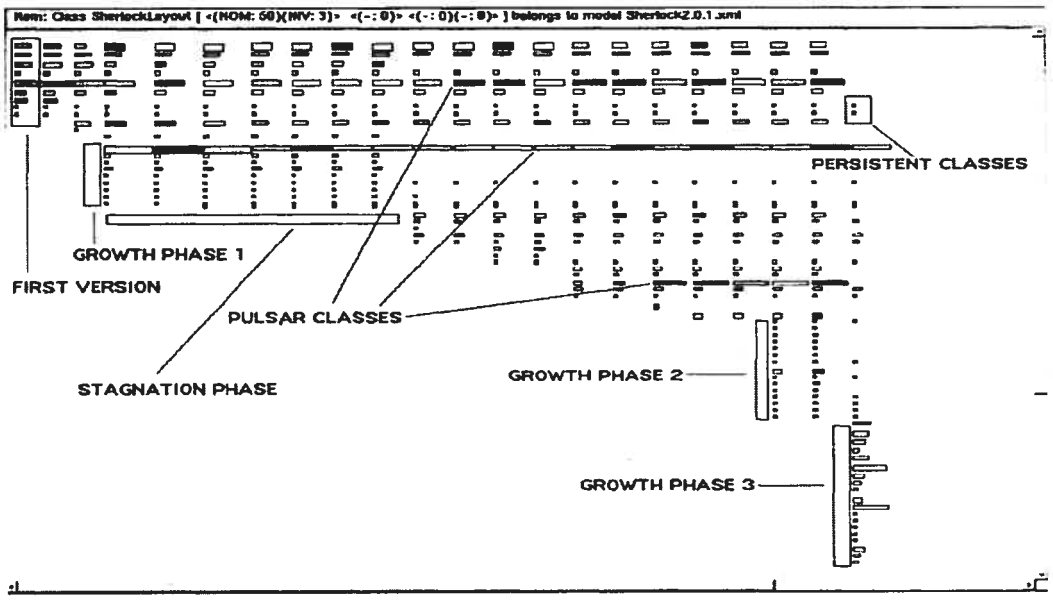


Figure 41 Vue globale de l'évolution du logiciel.

3.4 Conclusion

La plupart des outils de visualisation dans le domaine du logiciel offrent d'abord un grand support durant tout le cycle de vie du logiciel. Cela permet d'améliorer la qualité du logiciel en offrant une aide aux architectes système et aux concepteurs (section 3.3.1), aux développeurs et aux programmeurs (section 3.3.2) et finalement, aux testeurs (section 3.3.3). Par ailleurs, en combinant la visualisation des aspects statiques et des aspects quantitatifs, les outils de visualisation offrent aussi une assistance aux tâches de reverse engineering et de refactoring. Toutefois, peu de recherche a été effectuée dans le domaine de l'évaluation et la prédiction de la qualité du logiciel par la visualisation.

Chapitre 4

Analyse de la qualité par la visualisation

Nous commençons ce chapitre par la présentation de l'apport de la visualisation dans l'évaluation et l'estimation de la qualité du logiciel. Après, nous présentons les étapes nécessaires à toute visualisation qui sont la normalisation et l'association. En même temps, nous exposerons notre premier prototype de visualisation qu'on a nommé MDS.

4.1 Principe

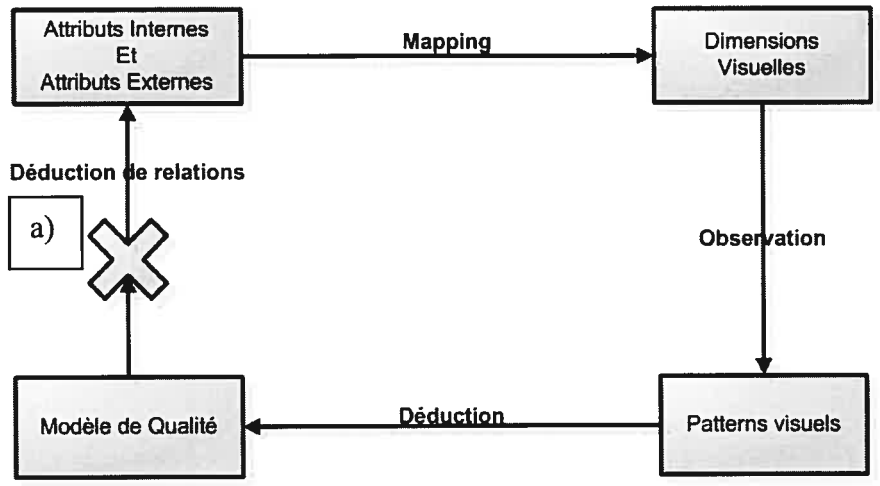


Figure 42 Modèle d'évaluation de la qualité par la visualisation.

L'évaluation et la prédiction de la qualité du logiciel se font en établissant des modèles qui permettent d'évaluer ses attributs externes, en identifiant leurs relations avec les attributs internes. Les attributs internes sont des attributs qu'on mesure directement et qui décrivent le produit logiciel indépendamment de son comportement. La taille, par exemple, est un attribut interne puisqu'elle peut être mesurée sans l'exécution du programme [53]. Par

contre, les attributs externes d'un logiciel sont les attributs qui ne peuvent être mesurés à priori. D'ailleurs, ils tiennent compte des relations du produit avec son environnement et ils portent sur son comportement. À cause de la faiblesse de la théorie dans le domaine de prédiction de la qualité, on ne peut directement déduire les relations entre les attributs internes et les attributs externes (voir figure 42).

Afin de remédier à ce problème, nous faisons un détour en commençant par faire une correspondance entre les attributs du logiciel et les dimensions visuelles. Par la suite, on procède à l'observation des patterns visuels afin de déduire des modèles de qualité. Tout d'abord, nous associons les attributs visuels avec les attributs quantitatifs internes du logiciel. Ensuite, on observe les patterns visuels qui pourraient coïncider avec les variations d'un attribut externe. Par exemple, si on associe la complexité à la valeur des abscisses X et la taille en KLOC (1 KLOC = 1000 lignes de code) à la valeur des Y, on peut alors spéculer que les classes constituant le regroupement en haut à droite seront « error prone » (attribut externe), ceci est infirmé ou confirmé en évaluant le nombre d'erreurs détectées dans ces classes. Par ce fait, on essaye d'établir une relation entre un attribut externe « error proness » et deux attributs internes, à savoir le nombre de lignes de code et la complexité. Cette manière est dite exploratoire. Elle se base surtout sur une hypothèse de départ qu'on essaye de raffiner en observant les patterns visuels (voir figure 43).

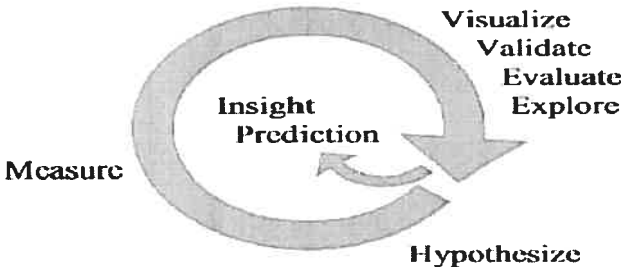


Figure 43 Modèle de visualisation scientifique exploratoire [52].

4.2 Détection de pattern visuel : réalisation

Dans un premier temps, nous avons réalisé un programme de visualisation générique, simple et flexible. Il est générique car il permet de présenter une entité quelconque et ces attributs, il est simple grâce à son modèle graphique intuitif et il est flexible au niveau de la correspondance entre les attributs visuels et les attributs des entités (du logiciel éventuellement).

4.2.1 Architecture générale

MDS est composé essentiellement de quatre modules. Un module de traitement de données StatUtil qui permet d'effectuer les calculs statistiques programmés avec MatLab. Un module de gestion de fichiers de données dont la fonction essentielle est la sauvegarde et le chargement de données. Un module d'association qui permet de faire l'association entre les attributs visuels et les attributs logiciels. Ce module utilise les interfaces JSwing de Java. Enfin, un module d'affichage implémenté en C ++, OpenGL et Qt 3.2 pour des questions de performance.

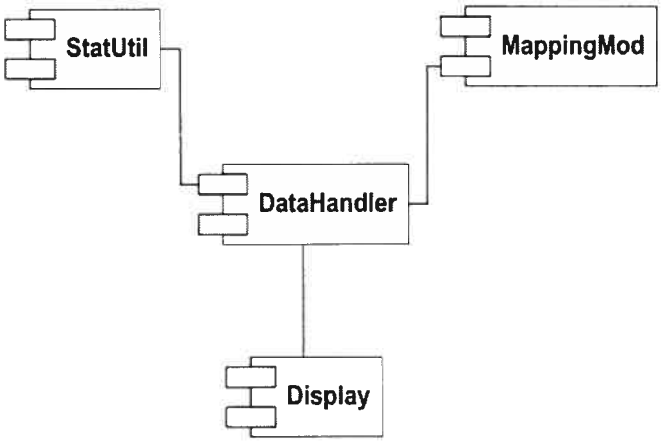


Figure 44 Architecture générale de MDV

4.2.2 Association entres les dimensions visuelles et les dimensions quantitatives

Nous savons que la combinaison entre les différents attributs visuels est la meilleure façon de véhiculer le plus d'information de manière efficace (voir section 2.5.1). Toutefois, il faut tenir compte des attributs « *preattentifs* », de leur importance relative et de leurs relations avec les attributs cognitifs (voir section 2.5.2).

MDV est un programme de visualisation flexible et indépendant du genre de données. Il permet de visualiser n'importe quel type d'entités et leurs caractéristiques. Ces entités pourraient être des classes, des modules où n'importe quels autres artéfacts. Cette flexibilité permet à l'utilisateur de définir son propre modèle de visualisation pour faire la correspondance entre les caractéristiques des attributs logiciel (ou autres) et les attributs visuels. Par ailleurs, les attributs visuels sont classés par ordre décroissant d'importance visuelle en partant du bas vers le haut comme le montre la figure 45. De cette manière, nous permettons à l'utilisateur d'exposer les attributs les plus importants et de mettre en arrière-plan les attributs qui sont de moindre importance. Par ailleurs, chaque dimension visuelle est associée à un seul attribut logiciel afin de créer une relation univoque entre eux.

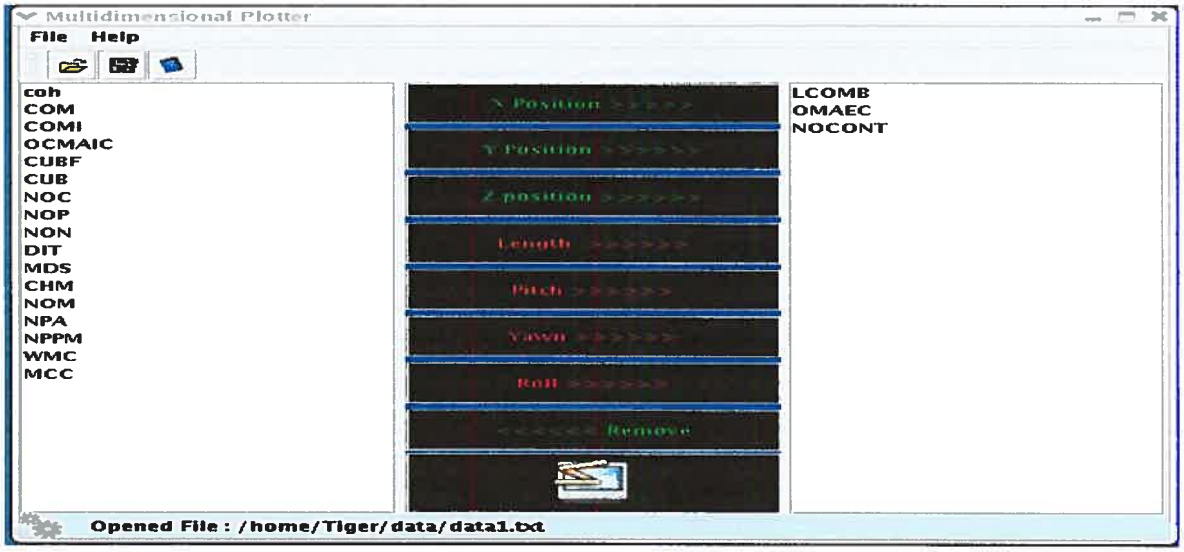


Figure 45 Interface de correspondance de MDV.

À gauche nous avons les dimensions existantes qui représentent des métriques. Au milieu il y a les attributs visuels, ceux qui sont verts c'est ceux qui ont déjà une correspondance avec des attributs internes. Finalement dans la partie le plus à droite, il y a les métriques choisies.

4.2.3 Modèle graphique

Le défi dans la construction du modèle graphique de MDV est celui de présenter le plus de dimensions possibles tout en produisant une visualisation qui permet d'explorer un volume important de donnés rapidement, sans effort et avec précision (section 2.2).

Puisque l'utilisation de la troisième dimension s'avère bénéfique peu importe le type de métaphore utilisée [56], nous avons donc décidé de développer un modèle tridimensionnel. Par ailleurs, ces trois dimensions visuelles présentant la position du modèle graphique sont des dimensions clés pour la détection des regroupements (cluster). Ainsi, ils sont placés dans le début de la liste des attributs visuels dans la fenêtre de correspondance (figure 45)

Ce modèle est en forme de flèche composé de trois parties (figure 46). Premièrement, en rouge, le cylindre dont la longueur présente une métaphore idéale pour des attributs quantitatifs [Tableau 3]. Deuxièmement, en bleu, le cône représentant le bout de la flèche, il permet de remarquer la rotation du modèle par rapport à l'axe des X et des Y (yawn, pitch) dans son système de coordonnées relatives. Troisièmement, en vert, le triangle présentant la plume de la flèche, permettant de distinguer la rotation du modèle autour de l'axe des Z (roll).

En résumé, les dimensions visuelles utilisées sont les suivantes :
$Va = \{ P_x, P_y, P_z, L, R_x, R_y, R_z \}$ représentant respectivement la position en x, y et z, la longueur du cylindre et la rotation en x, y et z.

Dû à l'hétérogénéité des données logiciel on ne peut pas faire une correspondance directe entre ses attributs et les attributs visuels. Un certain traitement s'impose qui est la normalisation de données.

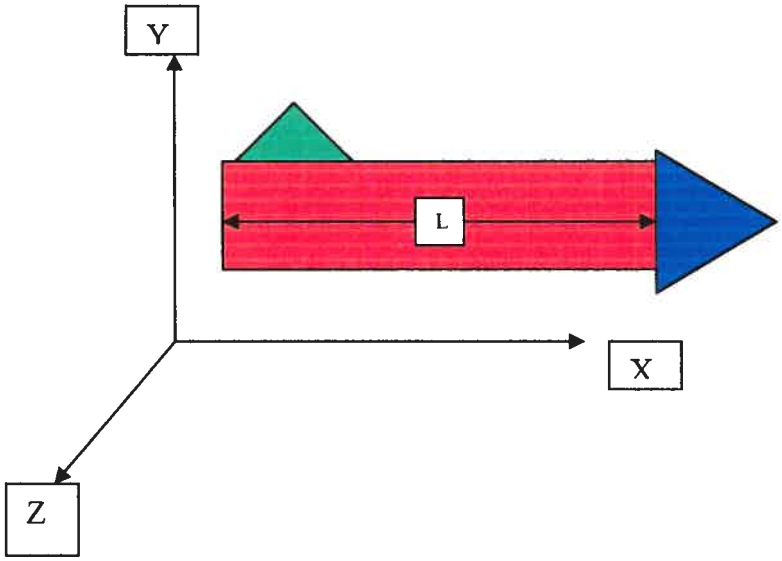


Figure 46 Description du modèle graphique. Nous avons 7 degrés de liberté : la position en X, Y et Z, la rotation autour des trois axes ainsi que la longueur de la flèche.

4.2.4 Normalisation

La normalisation est une transformation de données dans le but d'enlever l'effet de différence d'amplitude et d'échelle qui existent entre les attributs. Par exemple, la taille en nombre de lignes de code peut varier entre 0 et quelques milliers alors que le nombre d'attributs d'une classe tend à être beaucoup plus petit, par exemple entre 0 et quelques dizaines. Ainsi, si on fait une correspondance directe entre ses attributs et des attributs visuels comme respectivement les coordonnées X et Y, on aurait une présentation inutilisable plus les observations seront éparpillées sur une ligne droite. Donc, pour y remédier nous utilisons la normalisation qui nous permet non seulement d'enlever l'effet d'amplitude et d'échelle, mais aussi de les associer à un intervalle de valeurs raisonnables pour l'affichage et la manipulation. Par ailleurs, la normalisation doit répondre à quelques exigences.

4.2.4.1 Les qualités souhaitées d'une normalisation

Il est souhaitable qu'une normalisation ait les qualités suivantes : l'invariance à la translation, la multiplication et l'insensibilité aux valeurs extrêmes [57].

L'invariance à la translation signifie que l'ajout d'une valeur constante n'a pas d'impact sur les valeurs normalisées d'un attribut quelconque. Plus formellement, dénotons par N la fonction de normalisation et par A_i une valeur quelconque d'un attribut A.
$N(A_i) = N(A_i + c)$. Un exemple d'utilisation de la translation est l'ajout d'une constante positive à toutes les valeurs d'un attribut afin de se débarrasser des valeurs négatives. Chose nécessaire si on veut la faire correspondre à un attribut visuel qui ne peut pas prendre de telles valeurs, comme par exemple la longueur ou la largeur.

Par analogie, l'invariance à la multiplication signifie que la multiplication par une valeur constante différente de zéro de toutes les valeurs d'un attribut quelconque n'a pas d'impact sur sa normalisation, donc N (Ai) =N (Ai x C), $C \neq 0$. Une telle opération est souvent combinée avec la première pour transformer les valeurs d'un attribut entre deux valeurs fixées d'avance (exemple entre 0 et 1). Ces bornes sont utiles pour cerner la zone d'affichage.

En dépit du fait qu'elle est citée en dernier, l'insensibilité aux valeurs extrêmes ne demeure pas moins importante que les deux premières caractéristiques. Elle influence énormément la qualité de la visualisation, surtout lorsque ces valeurs constituent des valeurs clés de la fonction de normalisation. Un exemple : l'utilisation des valeurs minimale et maximale.

4.2.4.2 Quelques méthodes de normalisation

Il y a plusieurs méthodes de normalisation, nous citerons par la suite les plus connues dans le domaine et celles que nous avons utilisées.

La normalisation standard est le type de normalisation la plus simple. Elle est accomplie par une translation, en ajoutant –min, et une multiplication par 1/ (max - min) pour cerner les valeurs d'un attribut entre 0 et 1. Une telle normalisation est invariante à la translation et la multiplication. Toutefois, elle est très sensible aux valeurs extrêmes puisqu'elle utilise les valeurs minimale et maximale.

La normalisation SUM consiste à mettre le minimum à 0 en ajoutant –min comme la première méthode, mais en plus on divise par la somme de toutes les valeurs A_i. D'ailleurs, cette méthode est invariante à la multiplication et la translation. Elle est aussi plus robuste que la première méthode puisqu'elle est moins sensible aux valeurs extrêmes. Ceci est dû au fait qu'elle dépend seulement de la valeur minimale qui s'avère rarement

être une valeur extrême. Notons que les mêmes caractéristiques pourraient être obtenues en utilisant la moyenne à la place de la somme.

ZMUV est une méthode de normalisation qui a pour effet de mettre la moyenne à 0 en faisant une translation de (-Moyenne) et en divisant par la variance. Par ailleurs, celle-ci est aussi invariante à la translation et la multiplication. Toutefois, elle est encore plus robuste que les deux premières méthodes puisqu'elle ne dépend pas directement des valeurs minimales et maximales. Elle est donc moins sensible aux valeurs extrêmes.

Normalisation	**Méthode**
Standard	Ajouter (-min), diviser par (max-min)
SUM	Ajouter (-min), diviser par la somme.
ZMUV	Ajouter (-Moyenne), diviser par l'écart type.

[Tableau 1] Tableau récapitulatif des trois méthodes de normalisation [57].

Toutes ces transformations impliquent des opérations d'addition et de multiplication sur les valeurs d'attributs, mais est-ce que de telles opérations sont toujours permises, quel est leur impact sur la visualisation et ses interprétations?

4.2.4.3 Les différents types d'échelle : normalisation et association

Il ne faut pas perdre de vue que ces attributs logiciels sont des mesures qui ont été calculées selon une échelle définie d'avance. Donc, avant d'effectuer des opérations sur un attribut quelconque, il faut vérifier les transformations admissibles par son échelle. Il existe essentiellement cinq types d'échelles : nominale, ordinale, intervalle, ratio et absolue [53].

4.2.4.3.1 L'échelle nominale

L'échelle nominale définie une catégorisation primitive, elle est composée de différentes classes qui ne supposent ni ordre ni relation entre elles. D'ailleurs, n'importe quelle numérotation distinctive ou représentation symbolique est une mesure acceptable. En plus,

elle ne supporte pas la notion de norme et elle admet seulement les transformations une à une. Par exemple, dans un programme orienté objet nous distinguons trois types de classes : interface, abstraite, et autres. Cette classification ne suggère aucun ordre ni aucune amplitude. Par ailleurs, dans le contexte de visualisation on peut faire correspondre les différentes valeurs de cet attribut à des formes distinctives. Un exemple : associer la forme de cercle au type de classes interfaces, la forme de triangle aux classes abstraites et finalement, la forme de carré aux autres classes.

4.2.4.3.2 L'échelle ordinale

L'échelle ordinale se démarque de l'échelle nominale seulement par la notion d'ordre. Une telle notion implique la transitivité et la complétude. Elle accepte aussi toutes transformations utilisant une fonction monotone croissante, donc toutes les méthodes de normalisation citées ci-haut [Tableau 1] peuvent lui être appliquées. Cependant l'addition, la soustraction, ou tout autre opération arithmétique impliquant différentes catégories n'ont aucun sens. Par exemple, si on s'intéresse à la complexité des classes, on pourrait distinguer trois types de classes : simple, moyennement complexe et incompréhensible. Une telle classification implique nécessairement la transitivité puisqu'en ayant trois classes C1, C2 et C3 si C2 est plus complexe que C1 et C3 est plus complexe que C2, alors C3 est plus complexe que C1. Elle implique aussi la complétude puisqu'une classe ne peut être à la fois incompréhensible et simple. Un tel attribut pourrait être associé à la couleur, par exemple nous pouvons avoir trois couleurs vert, orange et rouge qui sont associées aux trois classes respectives. Ainsi une classe incompréhensible aurait la couleur rouge et une classe simple aurait la couleur verte.

4.2.4.3.3 L'échelle intervalle

Outre la préservation de l'ordre, l'échelle intervalle capture l'information à propos des intervalles qui séparent les différentes classes. D'ailleurs, elle accepte les opérations de multiplication et de soustraction. Toutefois, elle ne préserve pas les ratios donc le calcul de

la division ou la multiplication de deux classes n'a pas de sens. Ainsi cette échelle accepte toutes les transformations affines de forme ax + b. Par exemple la transformation des Fahrenheit aux degrés Celsius F= 9/5C+32.

4.2.4.3.4 L'échelle ratio

L'échelle ratio est l'échelle la plus connue, surtout dans le domaine physique. Elle préserve non seulement l'ordre et la taille d'intervalles entre les entités, mais aussi toutes les opérations de type ratio ou arithmétique. Par exemple la taille du code en KLOC est mesurée selon une échelle de type ratio, ainsi on peut dire que la taille d'un programme est deux fois plus grande qu'un autre. Une des meilleures correspondances qu'on peut faire entre ce genre d'attribut et un attribut visuel est celle avec la longueur ou la taille d'un symbole (exemple la longueur de la flèche).

4.2.4.3.5 L'échelle absolue

Pour finir nous citons l'échelle de type absolu, dont la mesure est faite seulement en comptant le nombre d'éléments. Un tel attribut prend toujours la forme du nombre d'occurrences de x dans l'entité. Par exemple le nombre d'attributs ou de méthodes dans une classe objet. Une telle échelle admet seulement les associations une à une et les transformations arithmétiques (addition, soustraction). Idéalement il faut associer un attribut de ce type à un attribut visuel de même type. Par exemple associer le nombre de méthodes dans une méthode au nombre de sommets ou de côtés dans un symbole graphique (Figure 47).

4.2.4.3.6 Association attributs de différentes échelles et attributs visuels

Nous remarquons que plus l'échelle est expressive plus elle est restreinte par rapport aux transformations [Tableau 2]. Idéalement, dans le processus de visualisation, il faut associer les attributs d'entités logicielles à des attributs d'entités visuelles ayant le même type d'échelle. Cependant, le nombre d'attributs visuels tend à être limité par rapport à celui des attributs d'entités logicielles, ce qui nous oblige souvent à faire des correspondances entre des attributs de types différents en passant par la normalisation. Une telle association requiert une prudence lorsqu'il s'agit de tirer des conclusions à propos de l'attribut associé à partir de sa représentation graphique. Par exemple, on peut faire la correspondance entre un attribut de type intervalle, comme la température, avec un attribut visuel de type ratio, comme la longueur d'un segment de droite. Cependant le fait de remarquer qu'un segment associé à un entier A est deux fois plus grand qu'un segment associé à une autre entité B, ne nous permet pas de dire que A est deux fois plus chaude que B puisque l'échelle de type intervalle ne permet pas les opérations de type ratio. Autrement dit, il faut respecter l'échelle la moins expressive.

Expressivité ↓ Restriction ↓

Échelle	**Transformation**	**Exemple de correspondance visuelle**
Nominale	Fonctions un à un	Forme
Ordinale	Fonctions monotones croissantes	Couleur
Intervalle	$F(x) = a\,x + b,\ a > 0$	
Ratio	$F(x) = a\,x,\ a > 0$	Longueur
Absolue	$F(x) = x$	Nombre de sommets/côtés

[Tableau 2] Tableau récapitulatif des différentes échelle et leurs transformations admissibles.

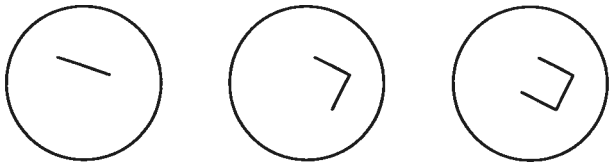


Figure 47 Le cercle représente une classe, le nombre de segments contenu représente le nombre de ses méthodes.

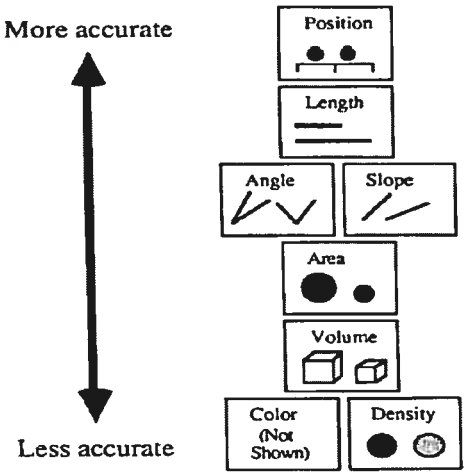


Figure 48 Des éléments visuels rangés par ordre croissant de leur efficacité de présentation des attributs quantitatifs (échelle ratio, intervalle, absolue) [58].

Quantitative	Ordinal	Nominal
Position	Position	Position
Longueur	Densité	Couleur tonalité
Angle	Couleur saturation	Texture
Pente	Couleur tonalité	Connexion
Surface	Texture	Densité
Couleur, saturation	Connexion	Couleur, saturation
Couleur, tonalité	Longueur	Forme
	Angle	Longueur
	Pente	Angle
	Surface	Pente
	Volume	Surface
		Volume

[Tableau 3] Classification des attributs visuels selon leur pertinence par rapport à l'échelle [58].

4.2.5 Réduction de dimensionnalité

Généralement le nombre d'attributs de données dépasse largement le nombre d'attributs visuels. Dans le cas d'un logiciel orienté objet, nous pouvons calculer des centaines de métriques reliées aux classes, des métriques qui tendent à être facilement calculables à partir du code ou autres. Cependant, au moment de la visualisation cette multitude d'attributs pose problème puisque les attributs visuels sont non seulement limités en nombre, qui sont fixés d'avance, mais en plus ils ont des importances différentes (section 2.5.2).

Les techniques de réduction de dimensionnalité permettent justement de diminuer le nombre d'attributs (dimensions) sans pour autant compromettre la structure ou le sens des données. La réduction de dimensionnalité est possible à cause de plusieurs raisons [59]. D'abord, beaucoup de paramètres (variables) ont une petite variance (par rapport aux autres variables) c'est-à-dire elles ont peu d'impact, donc on peut les éliminer sans changer la structure des données. Ensuite, il existe souvent des variables qui sont corrélées entre elles

(par exemple combinaison linéaire), ce qui implique une redondance qu'on peut éliminer en identifiant l'ensemble des variables indépendantes (non corrélées) qui représentent les dimensions intrinsèques.

Il existe plusieurs techniques de réduction de dimensionnalité qui permettent de réduire les dimensions tout en préservant l'information incarnée dans les données. En voici quelques unes : PCA (analyse de composante principales), MDS (multidimensional scaling), SOM (self-organizing maps), etc. Toutefois, ces techniques ont deux problèmes majeurs [60]. Premièrement, leurs algorithmes sont automatiques et n'impliquent aucune interaction avec l'utilisateur, ce dernier ne peut donc pas utiliser ses connaissances du domaine pour les améliorer. Deuxièmement, l'espace dimensionnel généré à partir des dimensions d'origine ne respecte pas nécessairement les transformations permises par les échelles et surtout il n'y a pas de sens intuitif pour l'utilisateur. De telles techniques permettent surtout d'observer les regroupements (clusters).

4.3 Conclusion

Le développement d'outils de visualisation efficaces et utilisables constitue un grand défi. D'abord, il faut établir un modèle graphique à la fois simple, clair et expressif. Ensuite, il faut pallier au problème de normalisation et d'association qui induit des erreurs d'interprétation surtout si on ne tient pas compte de la différence d'échelles.

Lors du développement du prototype (MDV) nous avons pris conscience de quelques problèmes qui touchent les présentations statiques tridimensionnelles. En premier, nous citons le volume des données qui cause un encombrement dans l'écran. Bien que nous avons ajouté des fonctionnalités de navigation il est toujours difficile de se situer dans un nuage de symboles (de flèches en occurrence). Deuxièmement, la troisième dimension implique une confusion lors de la comparaison des attributs et des symboles se situant dans des profondeurs différentes (sur l'axe des z). En effet, la projection perspective réduit la taille des symboles qui sont les plus profonds et accentue celle de ceux qui sont proches. Finalement, nous avons remarqué que les techniques de réduction de dimensionnalité sont

très efficaces pour classifier et différencier des objets ayant différents attributs. À titre d'exemple : discerner les différents chiffres (voir figure 49) et lettres dans un code postal. Cependant, les visualisations produites par ces techniques sont rarement utiles en raison de la disparition de l'information par rapport aux dimensions d'origine.

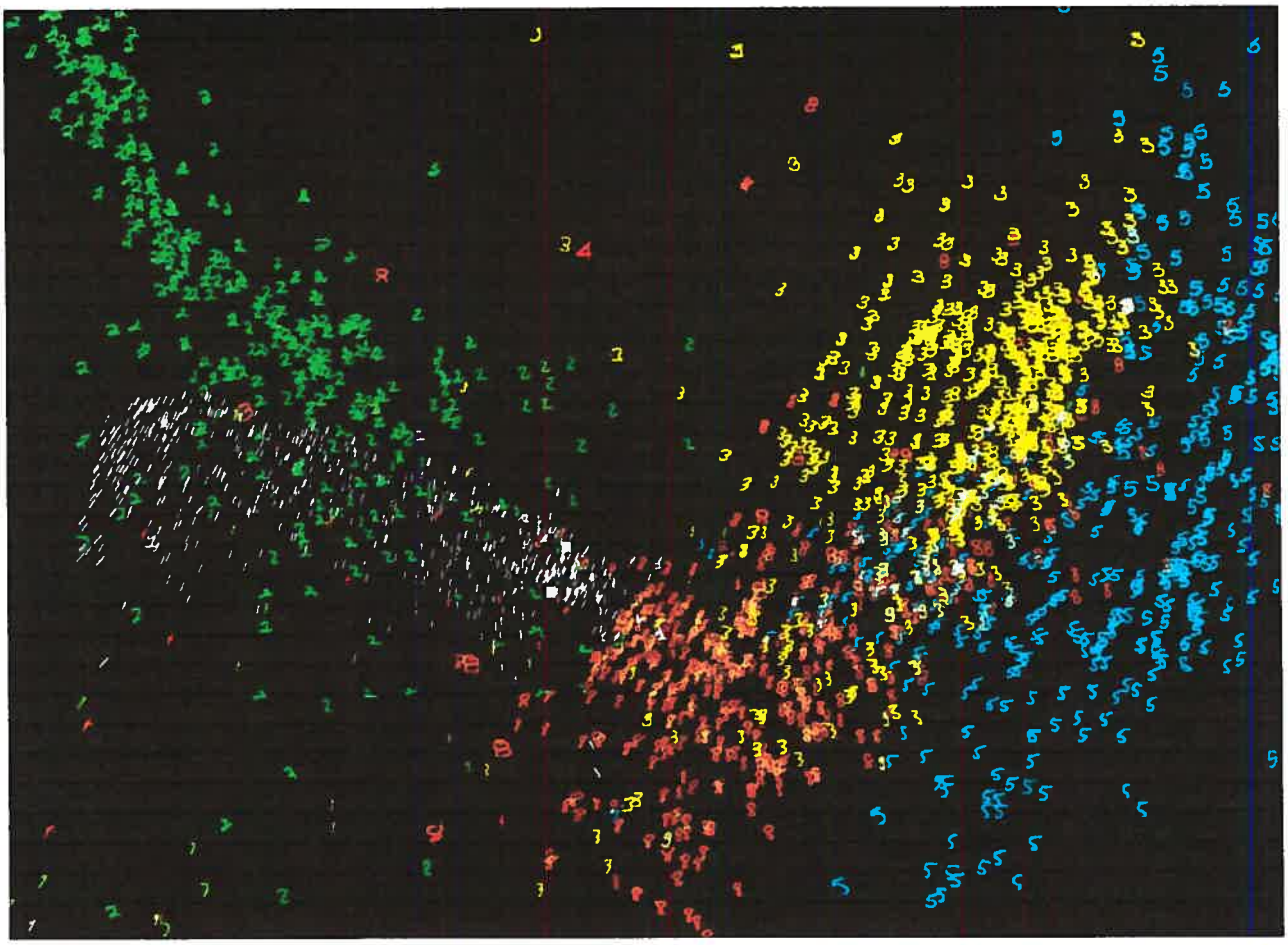


Figure 49 Une illustration de l'utilisation des techniques de réduction de dimensionnalité

Chapitre 5

Analyse de la qualité par simulation

Ce chapitre présente notre approche de simulation des phénomènes logiciels en se basant sur les modèles et les théories empruntées à d'autres domaines. Nous y présentons les différentes étapes requises pour la construction de telles simulations.

5.1 Introduction

Malgré des années de recherche et un nombre gigantesque d'applications pratiques, le développement et la maintenance du logiciel demeurent des activités très coûteuses et à haut risque. Le développement par les objets et les technologies apparentées a apporté beaucoup de progrès. En effet, ils ont permis de réduire de manière considérable le fossé entre les besoins et la vision des utilisateurs d'une part, et leur implémentation comme logiciel d'autre part. Cependant, dans la pratique, les coûts et le risque demeurent élevés comparés au développement et à la maintenance d'autres produits manufacturés [67][70]. Plusieurs raisons peuvent expliquer cet état de fait. La plus importante à notre avis est que de nombreux phénomènes liés au logiciel tels que l'évolution et la fiabilité sont complexes et très peu de théories existent pour les expliquer. Comprendre ces phénomènes et les modéliser sont des conditions essentielles à l'amélioration du contrôle sur les tâches de développement et la maintenance du logiciel.

Dans ce chapitre nous proposons une approche qui permet de réutiliser des modèles et des théories empruntées à d'autres domaines pour mieux comprendre les phénomènes liés au logiciel. Plus spécifiquement, l'idée est de partir d'une intuition de similarité entre un phénomène étudié et un phénomène connu dans un autre domaine, de prendre le modèle théorique correspondant à ce dernier, de l'adapter au contexte et aux contraintes du phénomène étudié et enfin de valider cette adaptation (figure 50).

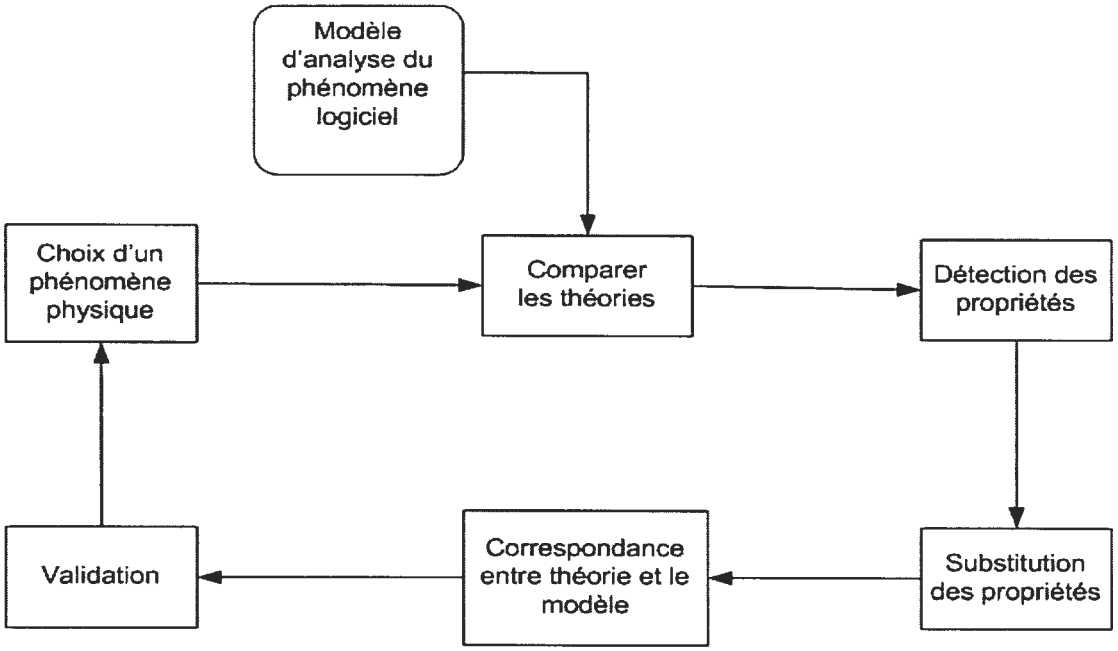


Figure 50 Processus de création du modèle de simulation

Dans ce qui suit nous allons présenter avec plus de détails ces différentes étapes. Ensuite nous exposerons trois exemples que nous avons étudiés et finalement nous allons présenter la plateforme que nous avons développé pour faire la simulation.

5.2 Processus de création du modèle de simulation

5.2.1 Le choix du phénomène physique

La plupart des visualisations et simulations s'accompagnent d'une légende décrivant la correspondance entre les attributs des données et les attributs visuels comme l'animation, les couleurs, etc. D'une manière générale ces présentations sont très efficaces. Néanmoins, leur clarté et facilité de compréhension sont compromises essentiellement à cause de deux facteurs. Premièrement, l'augmentation du nombre d'attributs impliquant la confusion et la désorientation de l'utilisateur. Deuxièmement, la correspondance entre les attributs visuels et les attributs des données est souvent vide de sens. En effet, la

sémantique des symboles (les attributs visuels) est très importante parce qu'elle touche directement l'aspect cognitif de la visualisation.

Le choix des phénomènes et des modèles d'autres domaines pour nous aider à comprendre les phénomènes liés au logiciel est basé surtout sur la similitude. À titre d'exemple nous pouvons voir qu'il y a une similarité entre le phénomène de propagation de la chaleur et le phénomène d'impact du changement dans le logiciel (voir Chapitre 6). Ou entre le phénomène d'attraction universelle qui stipule que « deux corps disposant d'une masse quelconque exercent une force d'attraction réciproque » et le phénomène de couplage entre les différentes classes. Il existe aussi une similitude entre le modèle d'atome proposé par le physicien Niels Bohr (voir figure 51) et le phénomène de cohésion des classes. Le modèle de Bohr présente les électrons comme étant des charges négatives en orbite autour du noyau sous l'influence du champ magnétique. Ces électrons sont sur différente couches de valences autour du noyau, plus on s'éloigne du noyau plus il est facile (en appliquant moins d'énergie) de les détacher [66].

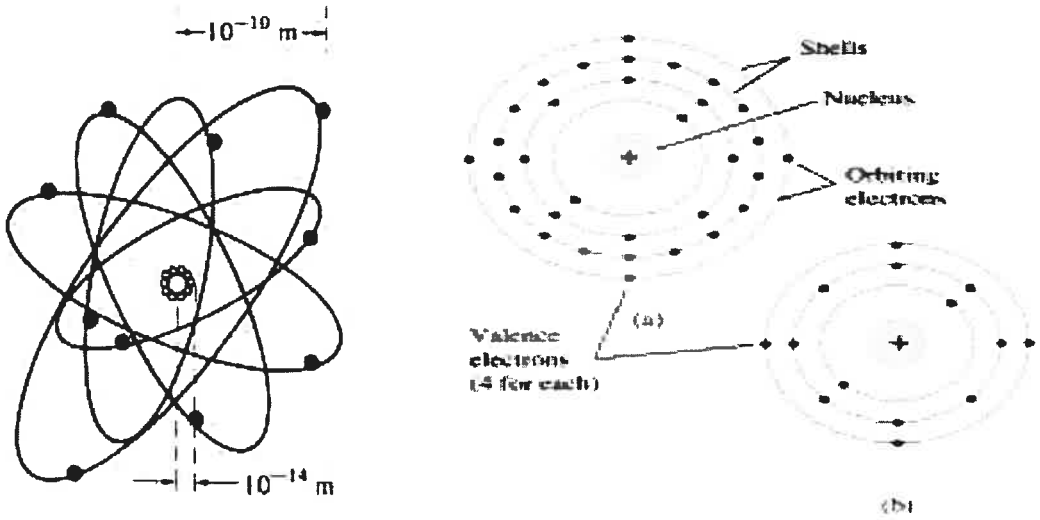


Figure 51 Modèle simple d'un atome.

L'utilisation des théories et des modèles empruntés à d'autres domaines pour expliquer des phénomènes logiciels présente plusieurs avantages. Elle nous permet d'utiliser la théorie et les lois des phénomènes qu'on maîtrise et qui sont déjà prouvés et validés. Ces derniers sont souvent reliés au domaine de la chimie ou la physique qui ont une très forte base mathématique qui permet d'établir des modèles de simulation rapidement et avec une certaine facilité. Le choix du bon modèle qui soit très similaire au phénomène logiciel étudié et dont nous avons une forte connaissance intuitive et de leur théorie favorise la détection plus rapide des incohérences.

5.2.2 Comparaison et détection des propriétés

En vue d'utiliser les lois et les modèles mathématiques qui régissent le phénomène emprunté il faut commencer par identifier ses caractéristiques les plus importantes. Ses attributs sont des déterminants centraux qui seront utilisés au moment de la correspondance entre le modèle logiciel et le modèle emprunté. Par ailleurs, l'identification de leur signification et leur rôle constitue une étape cruciale et déterminante pour le succès de la simulation.

À titre d'exemple nous aimerions bien faire une correspondance entre le modèle masse-ressort dans le domaine physique et le modèle de couplage dans le domaine logiciel. En effet ces deux phénomènes sont bien similaires du moins de par leur logique globale. La liaison de couplage entre deux classes pourrait être vue comme un ressort qui relie deux corps. De la sorte nous pouvons faire la correspondance entre les caractéristiques du couplage entre classes et les caractéristiques du ressort reliant deux corps.

Pour le modèle masse-ressort les caractéristiques les plus importantes sont la longueur L du ressort et sa raideur K (voir Figure 52), la masse et la densité de chaque corps pourraient être tenues en compte mais elles mettraient en jeu d'autres phénomènes comme la force gravitationnelle. Dépendamment du niveau de précision et de détails nous pouvons

chercher d'autres attributs du modèle choisi qui ont un impact indirect sur ses caractéristiques importantes. À titre d'exemple pour le modèle masse-ressort nous pouvons utiliser K comme constante ou la considérer comme une variable dépendante d'autres facteurs comme d'élasticité du matériau E (module d'Young), de la longueur L et de la section S du ressort.

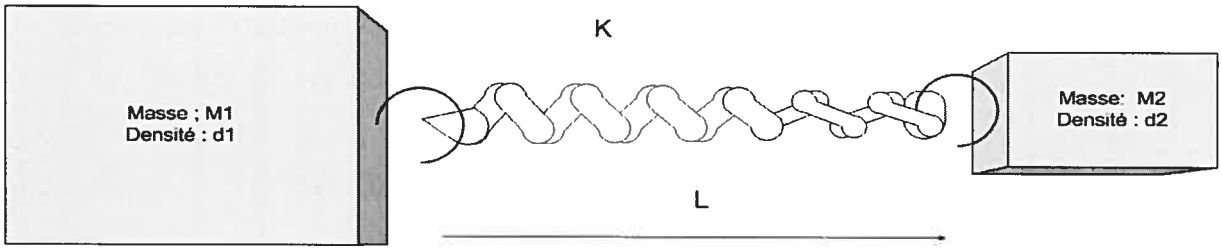
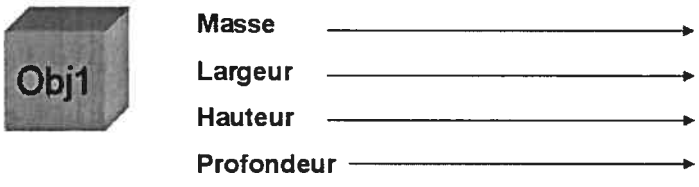


Figure 52 Schéma simplifié des attributs essentiel du modèle masse-ressort

5.2.3 Association et substitution

Pendant cette étape nous établissons des associations entre les attributs les plus pertinents de la théorie du modèle choisi et les attributs du phénomène logiciel. En effet, en reprenant l'exemple du système masse-ressort cité ci-haut, nous pouvons présenter les classes et leurs relations de couplage dans le monde logiciel par des corps et des ressort les reliant dans le modèle physique. Des métriques de classes (attributs des classes) sont ainsi associées aux attributs du corps (voir figure 53). Des métriques décrivant les relations entre les classes sont associées aux propriétés des ressorts reliant les corps qui les présentent. À titre d'exemple nous associons une métrique de couplage à la valeur de la raideur du ressort K et la valeur de la longueur L a une métrique d'héritage.



	C1	C2
WMC	125	65
NOM	6	3
NOA	6	12
NIM	1	2

Figure 53 Association entre les attributs d'un corps (parallélépipède) et des métriques de classe.

5.2.4 Adaptation du modèle physique et association entre les deux théories

Notre principal souci durant cette étape est de préserver la cohérence et la pertinence du modèle mathématique du phénomène emprunté. Pour ce faire nous utilisons les techniques de normalisation comme citées à la section 4.2.4. Cependant, les lois qui régissent les phénomènes logiciels et celle du domaine cible sont souvent de différente nature. En effet, une application directe des lois d'un domaine physique à un phénomène logiciel affecte le sens et le but même de la simulation.

Pour rapprocher les deux modèles nous effectuons certaines simplifications et ajustements de la théorie et du modèle mathématique du phénomène choisi tout en conservant sa logique globale. Par ailleurs nous avons eu recours à quelques modifications de la théorie de propagation de la chaleur afin de respecter certaines contraintes dans le monde logiciel (voir Chapitre 6).

5.2.5 Validation du modèle de simulation

Nous suggérons deux méthodes de validation, la validation par l'avis d'experts et la validation expérimentale. Les validations expérimentales sont utilisées dans une situation où l'on connaît déjà le résultat escompté de l'analyse du phénomène pour des cas particuliers et que l'on souhaite les comparer avec les résultats obtenus par le modèle de simulation. Cette méthode permet surtout d'améliorer le modèle de simulation en tenant compte des différences entre les lois et contraintes qui régissent le phénomène logiciel et celles qui régissent le phénomène choisi. En dernier lieu nous avons recours aux avis d'experts qui vont enrichir le modèle de simulation en suggérant certaines modifications inspirées de leurs expérience et expertises.

5.3 Conclusion

Le but même de notre étude est d'utiliser la visualisation et la simulation pour combiner les méthodes d'experts et les méthodes statistiques. L'utilisation des modèles de simulation empruntés à d'autres domaines pour expliquer certains phénomènes logiciels présente plusieurs avantages potentiels. Une telle approche améliore la compréhension des phénomènes logiciels en les associant aux modèles de phénomènes bien connus. L'exploitation de cette métaphore augmente considérablement notre capacité de détection des erreurs et des anomalies.

Chapitre 6

Modèle de propagation de la chaleur

Dans ce chapitre nous allons étudier le phénomène d'impact du changement dans le domaine logiciel en utilisant le modèle de propagation de la chaleur dans le domaine de la science physique. Par ailleurs, nous allons utiliser notre plateforme ANIMETRIX que nous avons développée pour faciliter les tâches d'association et de simulation.

6.1 Introduction

Dans ce chapitre, nous nous intéressons particulièrement au phénomène de la propagation des effets de changements dans le code durant l'évolution. En effet, lorsque le code doit être modifié pour corriger une erreur ou ajouter, modifier ou supprimer une fonctionnalité, la connaissance des effets des changements sur le système est un atout majeur. Elle permet de choisir parmi plusieurs alternatives en plus de réduire les coûts de tests en ne se concentrant que sur les parties affectées ainsi que de déterminer quand une restructuration majeure est nécessaire.

Cependant, une analyse exhaustive de l'impact de changements est impossible compte tenu des limites de l'analyse statique dans les langages à objet. Dès lors, des modèles d'approximation des impacts sont nécessaires. De nombreux modèles ont été proposés dans la littérature [68] [72] [74][77]. Malheureusement, ils n'offrent qu'une approximation grossière des impacts et ne modélisent le phénomène que de manière partielle.

Dans ce contexte, nous mettons en pratique l'approche proposée en utilisant un modèle de la propagation de la chaleur dans les systèmes physiques comme point de départ à la modélisation et la simulation de la propagation des effets de changements. Nous

commencerons par une description de la propagation des effets de changements en utilisant comme base le modèle décrit dans [68]. Nous introduirons ensuite le modèle de propagation de chaleur dans la section 6.3. L'adaptation de ce dernier au contexte et aux contraintes de la maintenance du logiciel est donnée dans la section 6.4. L'implantation du modèle résultant et son évaluation sur un exemple sont décrites dans la section 6.5. Nous conclurons ce chapitre par une discussion.

6.2 Analyse de l'impact de changements

De nos jours, les changements et les adaptations des logiciels sont de plus en plus fréquents. L'évolution de la technologie, les pressions économiques et les changements géopolitiques sont autant de raisons qui expliquent ce besoin constant de changements. Cette façon de faire est devenue à tel point la norme que l'on s'accommode aujourd'hui facilement de spécifications grossières et incomplètes pour commencer le développement d'un logiciel.

Ceci dit, le fait que les changements soient fréquents ne veut pas dire qu'ils sont faciles à effectuer. Au contraire, développer des modèles pour analyser leurs impacts et donc les contrôler est un souci constant de la communauté. Pour expliquer cette problématique, nous allons utiliser le modèle proposé dans [68] qui nous servir également pour l'adaptation (voir section 4).

6.2.1 Modèle d'impact du changement

D'après ce modèle, un système est vu comme un ensemble de classes reliées par différents types de liens. Tout changement dans une classe produit des effets qui se propagent à travers ces liens aux autres classes. Dans ce qui suit, nous présentons les différents types de changements et de liens ainsi que la notion d'impacts.

6.2.1.1 Types de changements

Toute modification complexe du code se traduit concrètement par un ensemble de changements élémentaires. :Les impacts de la modification est la conséquence des impacts des changements élémentaires qui la compose.

Les changements élémentaires peuvent être regroupés dans trois catégories : les changements faits sur une classe, sur un attribut ou sur une méthode. Les changements qui touchent la classe sont l'ajout, la suppression ou la modification de la structure d'une classe. Au niveau des méthodes, on trouve l'ajout ou la suppression d'une méthode et la modification de sa portée, de sa signature ou du type de retour. Finalement, pour les attributs, nous avons l'ajout et la suppression, mais également le changement de type ou de portée. Comme nous l'avons dit précédemment, tous ces changements sont considérés comme atomiques.

6.2.1.2 Types de liens

Les liens reflètent les types de relations qui peuvent exister entre les objets ou les classes. Ces liens sont en général indépendants les uns des autres et sont donc autant de véhicules des effets de changements. Le modèle considéré tient compte de l'association (S), de l'agrégation (G), de l'héritage (H) et de l'invocation (I). De plus, on considère le lien implicite entre une classe et elle-même (L). Pour chaque classe chaque lien détermine un ensemble des classes atteintes par ce lien. Par exemple, $S\ (c_i)$ est l'ensemble des classes reliées à c_i par un lien d'association.

6.2.1.3 Impact du changement

D'après le modèle considéré, l'impact direct d'un changement est l'ensemble des classes qui nécessitent une correction suite à ce changement. Cet ensemble varie selon le type de changement effectué et la nature des relations qu'a la classe concernée avec les autres classes. L'ensemble des classes impactées est calculé à partir des ensembles de base correspondant aux différents types de liens. Ainsi, pour un changement ch_j effectué sur une classe cl_i, l'impact sera

$$impact(cl_i, ch_j) = f(S(cl_i), G(cl_i), H(cl_i), I(cl_i), L(cl_i)) \quad [1]$$

Par exemple, si la portée d'une méthode d'une classe *cli* est changée (pm_1) de publique à protégée, l'ensemble des classes qui seront affectées est

$$impact(cl, pm_1) = I(cl) - H(cl) \quad [2]$$

où *I(cl)* est l'ensemble des classes dont les méthodes invoquent une méthode de *cl* et *H(cl)* les sous-classes de *cl.*

6.2.2 Modèle de propagation de la chaleur

La propagation de chaleur est l'échange de chaleur qui se produit entre deux corps de différentes températures. Ce phénomène se produit de trois manières : par convection, par rayonnement et par conduction. Nous nous intéressons uniquement à la propagation de la chaleur par conduction entre deux corps reliés par un connecteur qui permet le transfert de chaleur de l'un vers l'autre (figure 54).

Exposé à une quantité de chaleur dQ, un corps homogène possède essentiellement deux caractéristiques qui influencent l'augmentation de sa température : sa capacité spécifique de chaleur c et sa masse m. La constante c est spécifique à la matière qui compose le corps. Elle représente la quantité de chaleur (en Joule) requise pour changer la température du corps d'un degré Celsius. Ainsi, formellement la variation de la température dT d'un corps peut être décrite par l'équation suivante :

$$dT = \frac{dQ}{mc} \qquad [3]$$

Comme, nous pouvons le voir dans la figure 54, le transfert de la chaleur se fait par conduction à travers le lien qui relie les deux corps. Pour simplifier le modèle, nous supposons qu'il n'y a pas de perte de chaleur lors du transfert (lien isolé). La chaleur est transférée par collision des atomes du corps le plus chaud vers le corps le plus froid. Le transfert continue jusqu'à ce que le système atteigne un équilibre thermique dans lequel les deux corps ont la même température.

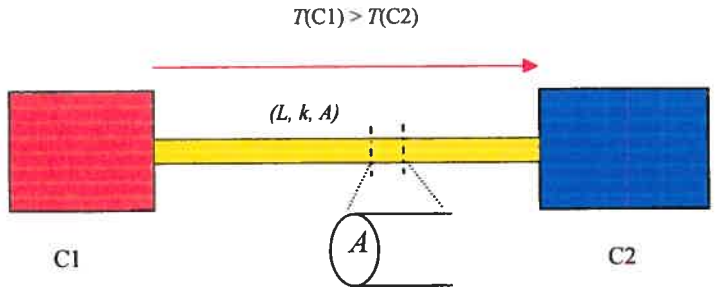


Figure 54 Principe du transfert de chaleur

Le taux de transfert de la chaleur par conduction dQ/dt (dt indiquant la période de temps) est proportionnel à la surface A de la section du lien, à sa conductivité thermique k et à la différence de température entre les deux corps. Toutefois, il est inversement proportionnel à la longueur L du lien. Ainsi, si on suppose que la température est uniforme tout au long du lien, le taux de transfert est défini par

$$\frac{dQ}{dt} = kA\frac{T(C1) - T(C2)}{L} \qquad [4]$$

L'équation 4 peut s'écrire aussi sous la forme :

$$\frac{dQ}{dt} = A\frac{(T(C1) - T(C2))}{R} \qquad [5]$$

avec $R = \frac{L}{k}$ qui représente la résistance thermique (inverse la conductivité).

6.2.2.1 Adaptation du modèle

Dans un logiciel à objets, l'application d'un changement à une classe peut nécessiter d'effectuer des changements dans les autres classes. Cet impact dépend d'une part, de la nature des liens et d'autre part des caractéristiques des classes elles-mêmes (complexité, taille, etc.). Parallèlement, comme nous l'avons vu dans la section précédente, dans un système physique en équilibre thermique, l'augmentation de la température d'un des corps résulte en un transfert de chaleur vers les autres corps. La température de ces corps change en fonction de leurs caractéristiques et des liens qui les relient. Intuitivement, on remarque une grande similitude entre les deux phénomènes. Ainsi, les classes ressemblent aux corps tandis que les liens entre classes sont similaires aux liens physiques entre les deux corps.
Nous avons donc naturellement pensé à utiliser le modèle physique pour la simulation de la propagation des effets de changements. Il est à noter, cependant, qu'il existe de nombreuses différences entre les deux phénomènes comme nous allons le voir dans le paragraphe suivant. Dans ce qui suit, nous allons d'abord décrire l'adaptation du modèle aux contraintes de la propagation des effets de changements. La deuxième étape est la détermination des différents paramètres du modèle.

6.2.2.2 Adaptation des lois

Bien que la similitude entre le phénomène logiciel et le phénomène physique soit grande, du fait des nombreuses contradictions, une application directe n'est pas possible. En effet, d'après les lois de la thermodynamique, l'augmentation de la température d'un corps implique un transfert d'une quantité de chaleur aux corps qui lui sont reliés. Le transfert se traduit donc par une diminution de la quantité de chaleur et donc de la température du corps

à l'origine du transfert. Dans le cas de changements dans une classe, la propagation des effets aux autres classes ne diminue pas l'impact local. En conséquence, le modèle est adapté de façon à ce que la chaleur transférée ne soit pas perdue.

Par ailleurs, dans un programme à objets, nous avons plusieurs types de changements. Les effets de chacun se propagent de manières différentes à travers les différents types de liens. Il est donc important de supposer que les paramètres des liens ne sont pas uniquement fonction du type de lien (S, G, H, I et L), mais également du type de changements. De ce fait, les propriétés des liens physiques telles que la conductivité varient en fonction du type de changement. Ainsi, dans le cas où un lien n'est pas concerné par un type de changement, on peut l'inhiber en affectant une très faible valeur à la conductivité du lien (ou une très grande valeur à la résistance).

6.2.2.3 Choix des paramètres

Comme nous l'avons vu dans la section 6.2.2, la propagation de chaleur dépend à la fois des caractéristiques des corps (m et c) et des caractéristiques des liens ($1/R$ et A). Nous avons déjà établi une correspondance entre classe et corps physique, entre lien et lien physiques et entre impact de changement et chaleur. Le but de cette étape est d'établir la correspondance entre les caractéristiques des corps et des liens physiques d'une part et celles des classes et des relations entre classes d'autre part.

Si on applique un changement à une classe cl_1, une classe cl_2 qui lui est reliée a d'autant plus de chances d'être affectée qu'elle est complexe [75]. Partant de ce constat, nous avons associé la capacité spécifique c à l'inverse de la complexité (calculée en utilisant la métrique WMC [69]. Nous considérons par ailleurs que la masse m est une constante pour toutes les classes. Nous obtenons ainsi

$$dT = \frac{dQ}{m} WMC \qquad [6]$$

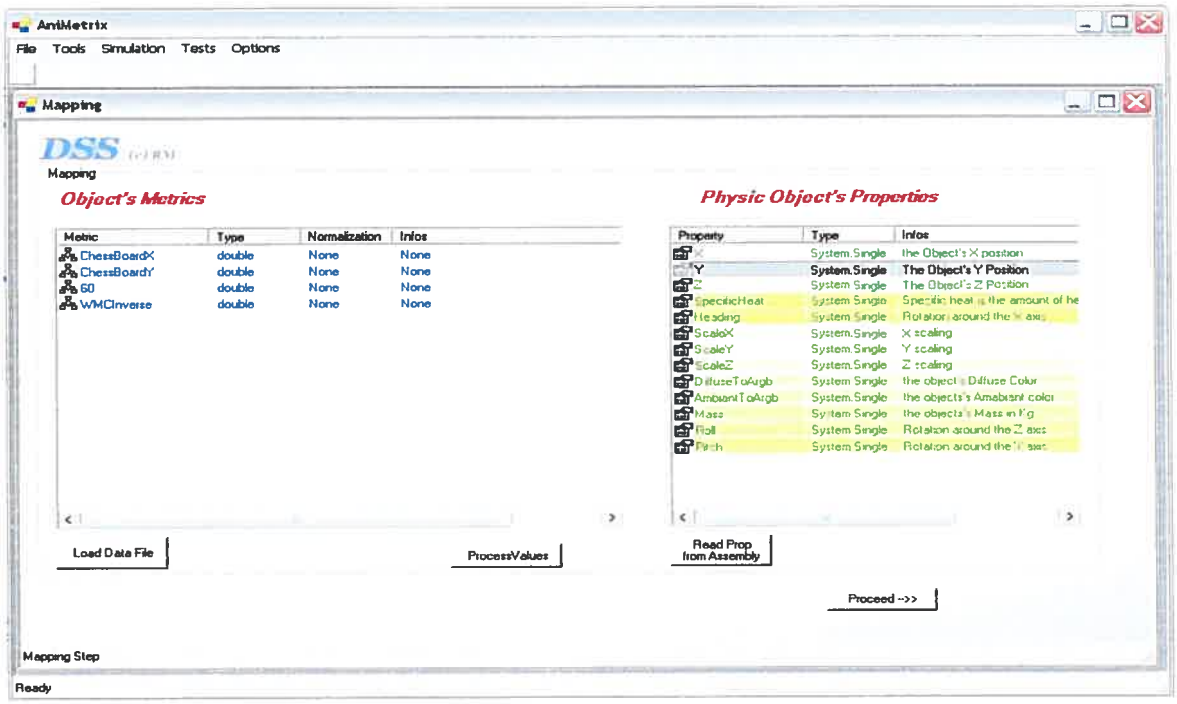
Du point de vue des liens, il est naturel de supposer que l'intensité des liens entre classes va favorisera la propagation des effets de changements. Par exemple, dans le cas du changement de portée d'une méthode d'une classe cl_1, le nombre de changements requis pour une classe cl_2 dépendra du nombre d'invocations à partir des méthodes de cl_2 de la méthode changée. L'intensité des liens dans les programmes à objets est calculée en général par la mesure du couplage en export entre deux classes. Nous associons donc une métrique de couplage en export CBOUsedBy et une variante de la métrique CBO [69] à la conductivité ($1/R$). Nous considérons par ailleurs que tous les liens ont la même surface de section A. Le taux de transfert est défini donc comme

$$\frac{dQ}{dt} = A \times CBOUsedBy \times (T(C1) - T(C2)) \qquad [7]$$

6.2.2.4 Implantation et exemple

Nous avons développé un outil appelé AniMetrix, permettant la visualisation et l'animation des données quantitatives des classes d'un programme à objets. AniMetrix permet de représenter un programme à objets sous la forme d'un graphe dans un espace tridimensionnel. Les nœuds du graphe représentent les classes et les liens, les relations entre classes. Les caractéristiques des liens et des nœuds peuvent être choisies respectivement parmi les métriques de classes ou de relations entre classes calculées au préalable. Pour la simulation, AniMetrix utilise les propriétés physiques des liens et des entités pour simuler des phénomènes physiques. Nous avons utilisé AniMetrix pour appliquer le modèle de propagation de la chaleur, mais également d'autres modèles comme celui des systèmes masse-ressort.

(a)



(b)

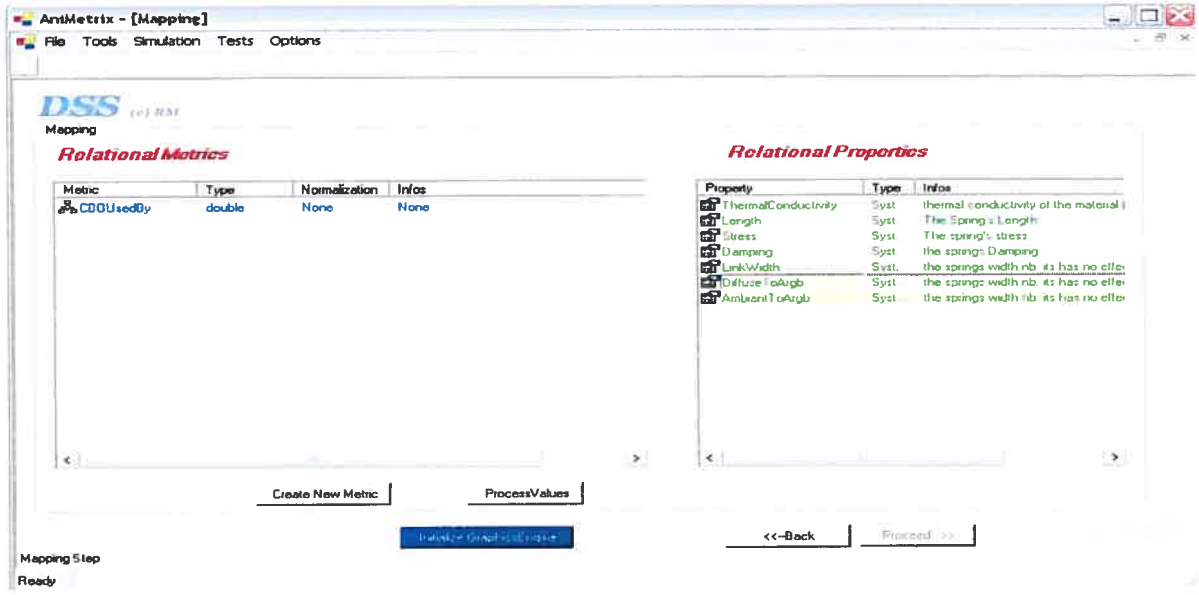


Figure 55 Définition des paramètres du modèle de simulation

Pour illustrer notre proposition, nous avons décidé d'évaluer notre modèle avec un système de petite taille GenExpert (72 classes). Ce choix est motivé par le fait que ce système a été développé dans notre équipe ce qui nous permet de vérifier facilement la pertinence des décisions prises par le modèle de simulation.

Comme nous pouvons le voir dans la figure 55, le choix des paramètres se fait par association de constantes ou de caractéristiques quantitatives des classes et des liens aux attributs des corps et des liens physiques. Ainsi, dans la figure 55.a, les coordonnées X et Y des corps sont associées à des fonctions de positionnement sur une grille, la profondeur Z à une constante et la capacité spécifique de chaleur à l'inverse de la métrique de classes *WMC*. Dans la figure 55 (b), seule la conductivité thermique est associée à la métrique de classes *CBOUsedBy*.

Figure 56 Représentation du programme à l'état initial

Une fois la simulation lancée, à l'état initial (voir figure 56), les classes du système sont présentées selon une grille. Toutes ont la même température de 0°C ce qui correspond à la couleur bleue. Le changement de température lors de la simulation se traduit par un changement de couleur selon un spectre allant du bleu au jaune en passant par le rouge (255°C). Pour permettre une meilleure visibilité, les liens ne sont pas affichés.

Lorsqu'on veut étudier l'impact d'un changement dans une classe, on augmente la température du corps correspondant à cette classe à 255°C (rouge). Dans la figure 57, on simule un changement dans la classe *ActionRule*. Suite à cette action le modèle est appliqué de manière récursive jusqu'à ce qu'aucune nouvelle propagation ne soit plus possible.

Figure 57 Simulation du changement dans la classe ActionRule

Selon le degré d'affectation, les classes vont changer de couleur. Comme nous pouvons le voir dans la figure 58, il y a plusieurs niveaux d'affectation qui correspondent à des couleurs différentes.

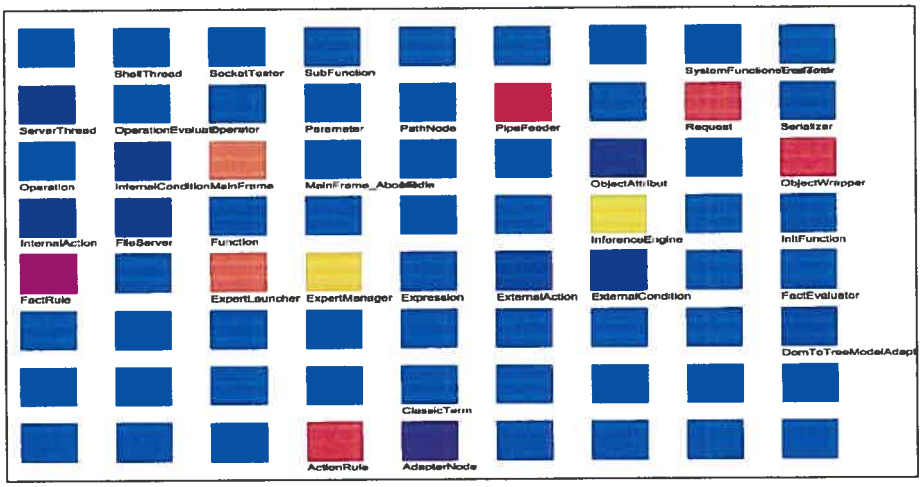


Figure 58 Propagation des effets du changement

Pour permettre une meilleure interprétation, il est possible d'appliquer un filtre qui permet de n'afficher que les classes affectées et dans l'ordre décroissant de leur degré d'affectation (voir figure 59). Ainsi, les classes de couleur jaune (ou proche du jaune) sont plus affectées que la classe qui a subi le changement. Parmi elles, on trouve la classe *ExpertManager*. Celle-ci est d'abord affectée directement car elle manipule une liste d'objets de type *ActionRule*. Elle est ensuite affecté indirectement car elle utilise les classes *InferenceEngine* et *ExpertLauncher* qui elles-mêmes sont affectées par les changements dans *ActionRule*.

Nous pouvons aussi remarquer qu'il y a des classes qui sont affectées autant que la classe changée (de couleur rouge). C'est le cas de *Request* par exemple qui est fortement couplée avec *ActionRule*.

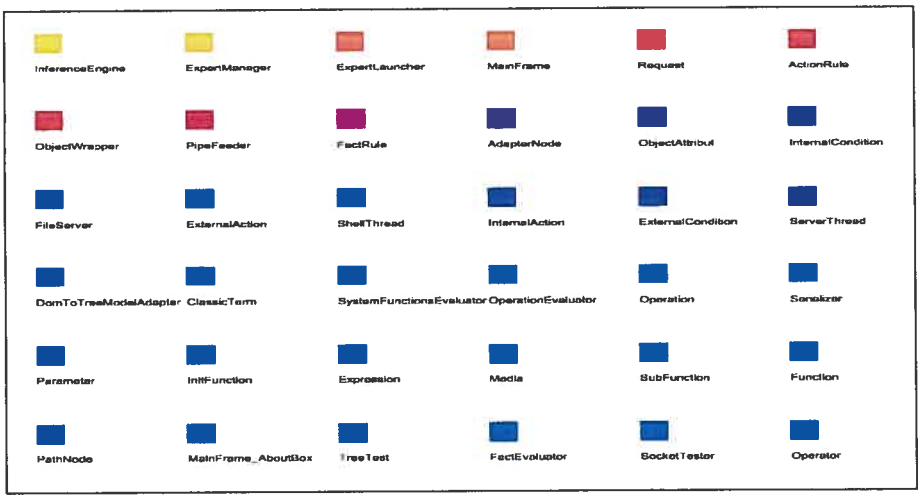


Figure 59 Ordonnancement des classes en fonction du degré d'impact

Finalement, nous distinguons des classes qui sont moyennement ou faiblement affectées (couleur tendant vers le bleu). Ces classes sont en général très faiblement couplées avec la classe changée comme *Operator*. Dans d'autres cas, ce sont des classes qui ne sont pas directement liées à la classe changée. Le lien comporte plusieurs niveaux d'indirections comme c'est le cas de *FrameAbout*. En effet, cette dernière fait partie de l'ensemble des classes formant l'interface utilisateur.

L'utilisation la plus important du résultat de la simulation est de définir l'ordre dans lequel on doit tester les classes suite au changement en suivant l'ordre défini dans la figure 59 Une autre utilisation du résultat de la simulation est de permettre de juger si le système ou une partie de celui-ci nécessite une restructuration majeure. En effet, si suite aux simulations de changement, la majorité du système ou d'un package est colorée de rouge à jaune, ceci indique des problèmes importants de conception.

6.3 Conclusion

L'utilité de recourir à des métaphores pour analyser le logiciel et comprendre les phénomènes qui influencent son cycle de vie est de plus en plus reconnue. De nombreux travaux montrent que cette approche facilite significativement l'analyse [73] [76] [71].

Dans cette étude, nous proposons d'aller encore plus loin en réutilisant, en plus de l'aspect statique, la dynamique des métaphores. En d'autres termes, nous proposons d'emprunter et d'adapter des modèles théoriques issus d'autres domaines pour modéliser et simuler des phénomènes liés au logiciel.

Nous illustrons cette position à travers l'exemple de l'adaptation du modèle de propagation de chaleur en thermodynamique à la modélisation et la simulation de la propagation des effets de changements dans les programmes à objets.

Chapitre 7
Conclusion

Le but de notre étude est de proposer une approche de visualisation et de simulation qui nous permet de mieux comprendre certains phénomènes liés au domaine du génie logiciel.

Nous avons commencé par introduire le système visuel humain. En premier lieu, nous avons mis l'emphase sur les limites et les capacités du système visuel humain. En second lieu, nous avons présenté les différentes méthodes de stimulation et de contrôle de l'attention. Nous avons conclu que le développement d'un bon système de visualisation et de simulation efficace a entre autres deux préalables essentiels. D'abord, il faut tenir compte des limites et des capacités du système visuel humain. Ensuite, il faut utiliser la combinaison de plusieurs stimuli facilitant ainsi la différenciation et la reconnaissance des symboles.

Nous avons proposé notre approche de simulation qui permet de réutiliser des modèles et des théories empruntées à d'autres domaines pour comprendre les phénomènes liés au logiciel. En effet, de nombreux travaux montrent que cette approche facilite significativement l'analyse [73] [76] [71]. Toutefois, nous nous sommes démarqués de ces travaux en adaptant les modèles de simulations dynamiques pour visualiser certains phénomènes du domaine du génie logiciel.

En vue d'implémenter notre approche, nous avons développé un outil appelé AniMetrix. Cet outil permet en premier lieu, de faire les associations entre les caractéristiques des entités logicielles (des métriques) et les attributs des éléments du modèle empruntés un autre domaine. En second lieu, il permet de simuler le phénomène choisi en temps réel. L'utilisateur peut intervenir pendant la simulation en changeant les caractéristiques des entités qui entrent en jeu et observer les résultats instantanément. Cet outil possède des fonctionnalités de navigation et de positionnement dans l'univers à trois dimensions. On

peut aussi à tout moment utiliser les fonctionnalités de capture d'écran ou d'enregistrement d'un film de la simulation. Cet outil est flexible et extensible. Flexible, en permettant le changement des associations et des méthodes de normalisation en tout temps. Extensible, puisqu'en implémentant les interfaces préalables nous pouvons ajouter d'autres symboles graphiques tridimensionnels.

Nous avons conclu cette étude en utilisant notre approche pour étudier le phénomène d'impacts du changement dans le domaine logiciel, en utilisant le modèle de propagation de la chaleur dans le domaine de la science physique. Pendant cette partie nous avons décrit avec détails le phénomène physique et le phénomène logiciel. Ensuite nous avons procédé à l'adaptation du modèle physique pour respecter les équivalences et différences dans monde logiciel. Finalement, nous avons effectué la simulation en utilisant Animetrix pour détecter les impacts du changement dans une application que nous avons développée.

Les idées et les exemples présentés ici sont encore à un stade embryonnaire. Un perfectionnement et une validation rigoureuse sont nécessaires pour tirer des conclusions finales. Cependant, nous croyons que ce travail donne une perspective nouvelle à l'étude de certains phénomènes logiciels.

Dans nos futurs travaux nous comptons étendre les fonctionnalités d'AniMetrix en ajoutant la possibilité de visualisation à différents niveaux de granularité. Nous voulons aussi appliquer notre approche pour simuler et visualiser d'autres phénomènes logiciels, à titre d'exemples l'utilisation de phénomènes d'attraction universelle pour simuler le phénomène de couplage entre les différentes entités logicielles.

Références

[1] Robert P. Bosch, Jr. "Distributed Algorithm Visualization" http://www-flash.stanford.edu/~bosch/cs348c/

[2] IBM Research Center http://www.cs.arizona.edu/icon/progvis/lectures/intro.htm

[3] Teresa Hübscher-Younger, N. Hari Narayanan "Dancing Hamsters and Marble Statues: Characterizing student Visualization of Algorithms". Proceedings of the 2003 ACM Symposium on Software Visualization, pages 95-104

[4] Alexander A. Sherstov "Distributed Visualization of Graph Algorithms", ACM SIGCSE Bulletin , Proceedings of the 34th SIGCSE technical Symposium on Computer Science Education, Volume 35, Issue 1, 2003.

[5] Stephen G. Eick, Eric E. et Summer Jr "A Tool for visualizing line oriented software statistics" IEEE Transactions on Software Engineering, Volume 18, Issue 11, 1992.

[6] Andrian Marcus, Louis Feng et Jonathan I. Maletic "3D Representations for Software Visualization" Maletic Proceedings of the ACM Symposium on Software Visualization, 2003.

[7] Eick S., Stefeen J. L., et Summer E. E. "Seesoft - A Tool For Visualizing Line Oriented Software Statistics". IEEE Transactions on Software Engineering, volume 18, 1992.

[8] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II "End-User Software Visualizations for Fault Localization" Proceedings of the 2003 ACM symposium on Software visualization , Pages: 123 - 132

[9] Gregg M. Townsend "Some Tools for Visualizing Icon Programs" 1997

[10] Loe Feijs et Roel De Jong, "3D Visualization of Software Architectures" Communications of the ACM, Volume 41, Issue 12 Pages 73-78 1998.

[11] Markus Eiglsperger, Michael Kaufmann et Martin Siebenhaller "A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams" , Proceedings of the 2003 ACM Symposium on Software Visualization, page 189.

[12] Glossaire http://www.pavevision.org/glossary.htm

[13] Ed H. Chi "A Framework for Visualizing Information" Kluwer Academic publishers, 2002.

[14] David P. Tegarden "Business Information Visualization" Communications of the AIS Volume 1, Issue 1 Article N 4, 1999.

[15] Dictionnaire Larousse Englais-Englais

[16] Wolfe, J.M.. "Guided Research 2.0 A Revised Model of Visual Search" Psychonomic Bulletin and Review, Vol.1, No.2, p.202-238, 1994.

[17] Laura G. TATEOSIAN "Nonphotorealistic Visualization of Multidimensional Datatsets", 2001.

[18] Donald B. Lindsley "Cortical Functions and Their Measurement: Vision as a Prototype", 2003

[19] David P. Tegarden, "Business information Visualization" Communications of the AIS Volume 1, Issue 1, Article No. 4 1999.

[20] George A. Miller "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", The Psychological Review, vol. 63, pp. 81-97 1956.

[21] Victor Ostromoukhov "Efficient Color For Efficient Web Applications" 2003.

[22] Jörn Kohlhammer, David Zeltzer "Towards a Visualization Architecture for Time-Critical Applications" Proceedings of the 9th international conference on Intelligent user interface p 271-273, 2004.

[23] Christopher G. Healey and James T. Enns "Large Datasets at a Glance: Combining Textures and Colors in Scientific Visualization" IEEE transactions on Visualization and Computer Graphics pp 145-167, 1999.

[24] Derrick Parkhurst and Ernst Niebur "Salience and Overt Visual Attention" 2004.

[25] Jan Theeuwes "Top-down search strategies cannot override attentional capture", Psychonomic Bulletin & Review, Volume 11, Number 1, 1 pp. 65-70(6) February 2004.

[26] Charles L. Folk, Andrew B. Leber and Howard E. Egeth, "Made you blink! Contingent attentional capture produces a spatial blink" Perception and Psychology 64(5):741-53, 2000.

[27] Howard E. Egeth and Steven Yantis "VISUALATTENTION: Control, Representation, and Time Course" Annual Psychological Revue 48:269–97, 1999

[28] Folk, C. L., Remington, R. W., & Johnston, J. C. "Involuntary covert orienting is contingent on attentional control settings", Journal of Experimental Psychology Human Perception Performance 18(4):1030-44. (1992)

[29] Theeuwes, J. "Perceptual selectivity for color and shape", Proceedings of the Twenty First Annual Conference of the Cognitive Science Society (pp. 649-652) 1992.

[30] Itti, L., & Koch, C. "Saliency based search mechanism for overt and covert shifts of visual attention", Vision Research. 40(10-12):1489-506, 2000.

[31] Theeuwes, J. "Stimulus-driven capture and attentional set: Selective search for color and visual abrupt onsets", Journal of Experimental Psychology Human Perception Performance, 20(4):799-806, 1994.

[32] Bacon, W. F., & Egeth, H. E. "Overriding stimulus-driven attentional capture" Journal of Perception and Psychophysics, 55(5):485-96. 1994.

[33] Theeuwes, J., & Burger, R. "Attentional control during visual search: The effect of irrelevant singletons", Journal of Experimental Psychology Human Perception Performance 24(5):1342-53, 1998.

[34] Christopher G. Healey "Perceptual Techniques for Scientific Visualization" 2001

[35] Stasko, J., Badre, A. and Lewis, C "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis" Proceedings of INTERCHI'93 Conference on Human Factors in Computer Systems, Amsterdam, pp. 61-66, April 1993.

[36] Thomas L. Naps, James R. Eagan, Laura L. Norton, "JHAVI -- An Environment to Actively Engage Students in Web based Algorithm Visualizations", ACM SIGCSE Bulletin , Proceedings of the thirty-first SIGCSE technical Symposium on Computer science education, Volume 32 Issue 1, 2000.

[37] Gruia-Catalin Roman et Kenneth C. Cox "Program Visualization: The Art of Mapping Programs to Pictures", Proceedings of the 14th International Conference on Software Engineering , pages 412-420, 1992.

[38] Blaine A. Price, Ronald M. Baecker et Ian S. Small "A Principled Taxonomy of Software Visualization" Journal of Visual Languages and Computing 4(3):211-266, 1994.

[39] Andrian Marcus,Louis Feng et Jonathan I Maletic. "3D Representations for Software Visualization" Proceedings of the ACM Symposium on Software visualization 2003.

[40] Martin Fowler "UML Distilled: A Brief Guide to the Standard Object Modeling Language", Addison-Wesley Professional 2 ème Edition, 1999.

[41] Oliver Radfelder et Martin Gogolla "On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation", Proceedings of the Working Conference on Advanced Visual Interfaces, 2000.

[42] T. Dwyer, "Three dimensional UML using force directed layout", Australian Symposium on Information visualisation, 2001.

[43] Girish Parikh "Exploring the world of software maintenance" ACM SIGSOFT Software Engineering Notes, Volume 11, Issue 2, 1986.

[44] Shneiderman, B. "Tree Visualization with TreeMaps: A 2-D space-Filling approach. ACM Transactions on Graphics (TOG)", Volume 11 Issue 1, 1992.

[45] Stephen T. Albin "Art of Software Architecture: Design Methods and Techniques" Addison-Wesley, 2003.

[46] Stephen H. Kan "Metrics and Models in Software Quality Engineering", Addison-Wesley, 2002.

[47] Gregory Swanson et Lee Globus "Software Metrics Visualization For a Graphical Programming Environment" 2001.

[48] Rainer Koschke "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey", Journal of Software Maintenance: Research and

Practice, Volume 15, Issue 2, 2003.

[49] Alexandru Telea, Alessandro Maccari, Claudio Riva "An Open Toolkit for Prototyping Reverse Engineering Visualizations", Proceedings of the Symposium on Data Visualisation, 2002

[50] Michele Lanza, Stéphane Ducasse "A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint", ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications, Volume 36, Issue 11, 2001.

[51] Michele Lanza "The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques", Proceedings of the 4th International Workshop on Principles of Software Evolution 2001.

[52] Robert M. Kirby, Daniel F. Keefe et David H. Laidlaw "Painting and Visualization" 2003.

[53] Norman E. Fenton et Shari Lawrence "Software Metrics" Course Technology 1997

[54] C. Ravindranath Pandian "Software Metrics: A Guide to Planning, Analysis, and Application", Auerbach Publications 2004.

[55] David H. Laidlaw "Scientific Visualization Research Issues" NIH/NSF Workshop on Visualization Research Challenges, 2004.

[56] C. Russo Dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud "Mapping Information onto 3D Virtual Worlds", Proceedings of the International Conference on Information Visualisation page 379, 2000.

[57] Mark Montague et Javed A. Aslam "Relevance Score Normalization for Metasearch" Proceedings of the Tenth International conference on Information and Knowledge Management, 2001.

[58] Jock Mackinlay "Automating the Design of Graphical Presentations of Relational Information" ACM Transactions on Graphics (TOG), Volume 5, 1986.

[59] Miguel Á. Carreura-Perpiñán "A Review of Dimension Reduction Techniques" 1997.

[60] J. Yang, M.O. Ward, E.A. Rundensteiner et S. Huang "Visual Hierarchical Dimension Reduction for Exploration High Dimensional Datasets", in Proceedings of the Symposium on Data Visualization VISSYM '03, 2003.

[61] Pollack, I J. Acoust Soc. Amer "The Information of Elementary Auditory Displays", The Journal of the Acoustical Society of America Volume 26, Issue 2, pages 155-158 1954.

[62] Hake, H.W., et Garner, W.R "The effect of presenting various numbers of discrete steps on scale reading accuracy", J Exp Psychol. Volume 42, pages 358-66. 1951.

[63] Simons, D. J., & Rensink, R. A. "Induced failures of visual awareness". Journal of Vision, 3(1), 2003 http://journalofvision.org/3/1/1x/

[64] J. Acoust et Pollack, I. "The information of elementary auditory displays", The Journal of the Acoustical Society of America Volume 26, Issue 2, pages 155-158 1954.

[65] Alessandro Orso, James Jones et Mary Jean Harrold "Visualization of Program-Execution Data for Deployed Software", 2003.

[66] Robert L. Boylestad et Louis Nashelsky "Electronics Devices and Circuit Theory", Prentice Hall, 2001.

[67] D. Bell. "Software Engineering, A Programming Approach" , Addison-Wesley, 2000.

[68] M. A. Chaumun, H. Kabaili, R. K. Keller et F. Lustman. "A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems". In Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering, 1999.

[69] S.R. Chidamber et C.F. Kemerer. "A Metrics Suite for Object-oriented Design". Rapport technique E53-315, MIT Sloan School of Management, 1993.

[70] D. Hamlet et J. Maybee. "The Engineering of Software", Addison-Wesley, 2001.

[71] C. Knight et M.C. Munro. "Virtual but Visible Software". Dans Proc. International Conference on Information Visualisation, 2000.

[72] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima et C. Chen, "Change Impact Identification in Object Oriented Software Maintenance" . Proceedings of the International Conference on Software Maintenance, pages: 202 - 211, 1994.

[73] M. Lanza et S. Ducasse. "Understanding Software Evolution using a Combination of Software Visualization and Software Metrics", Conférence sur les langages et modèles à objets, 2002.

[74] L. Li et A. J. Offutt. « Algorithmic Analysis of the Impact of Changes to Object-Oriented Software". In ICSM96, 1996.

[75] W. Li et S. Henry, "Object-Oriented Metrics that predict Maintainability", Technical Report: TR-93-0, 1993.

[76] T. Panas, R. Berrigan et J.C. Grundy. "A 3D Metaphor for Software Production Visualization". Dans Proc. International Conference on Information Visualisation, 2003.

[77] Václav Rajlich, "A Model and a Tool for Change Propagation in Software", ACM SIGSOFT Software Engineering Notes archive, Volume 25, Issue 1, page:74, 2000.